

Collaborative Software Exploration with the Oculus Rift in ExplorViz

Bachelor's Thesis

Daniel König

September 29, 2018

KIEL UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE
SOFTWARE ENGINEERING GROUP

Advised by: Prof. Dr. Wilhelm Hasselbring
M.Sc. Christian Zirkelbach

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Kiel,

Abstract

The complexity of software projects is increasing and so is the difficulty of understanding them. Modern software visualization software can be used to support this understanding. It makes it possible to comprehend software not only at a code level, but with the help of visual illustrations. Due to technological progress, software landscapes today can not only be viewed visually on a monitor. The state-of-the-art technology called virtual reality offers a different and immersive experience. It allows users to dive into the software using devices such as the Oculus Rift or HTC Vive.

This thesis presents an approach that allows the exploration of software landscapes in virtual reality collaboratively. The software visualization tool ExplorViz is extended for this purpose. The existing VR plugin is further developed and a multi-user functionality is realized with the help of WebSocket connections. Since software is usually developed in a team, members of a team can come together in the virtual world. To the best of our knowledge, there is no comparable approach at this time.

Through an evaluation we verified the usability of our approach. For this 11 groups with 2 persons each tested and rated our collaborative VR approach. The results indicate a positive experience and good usability. Especially well rated was the impression that the participants felt to be in the same virtual world.

Contents

1	Introduction	1
1.1	Motivation	1
1.2	Goals	1
1.2.1	G1: Concept/Design	1
1.2.2	G2: Implementation	2
1.2.3	G3: Evaluation	2
1.3	Document Structure	2
2	Foundations and Technologies	3
2.1	Virtual Reality	3
2.2	ExplorViz	4
2.3	Three.js	7
2.4	ExplorViz VR Extension	8
2.4.1	Application and Landscape View	8
2.4.2	Interaction	8
3	Concept/Design	11
3.1	General Concept	11
3.2	Architecture	12
3.3	Frontend	13
3.3.1	Landscape and Applications	13
3.3.2	User Visualization	14
3.3.3	Options Menu	14
3.3.4	User List	16
3.3.5	Overlays	16
3.4	Backend	17
4	Implementation	19
4.1	Frontend	19
4.1.1	WebSockets	19
4.1.2	Landscape and Applications	20
4.1.3	Teleportation	23
4.1.4	Menus	23
4.1.5	User Visualization	28
4.1.6	Controls	30
4.2	Backend	31

Contents

4.2.1	Main Structure	31
4.2.2	Main Loop & Messages	32
4.2.3	Synchronization	32
5	Evaluation	33
5.1	Goals	33
5.2	Methodology	33
5.3	Experiment	33
5.3.1	Survey	33
5.3.2	Experimental Setup	37
5.3.3	Experiment Procedure	38
5.4	Results	39
5.4.1	Test Subjects	40
5.4.2	Interaction Results	41
5.4.3	General Results	41
5.4.4	Visual Appearance Results	42
5.4.5	Spectate Feature Results	42
5.5	Discussion	43
5.5.1	Interaction Results	43
5.5.2	General Results	43
5.5.3	Visual Appearance Results	43
5.5.4	Spectate Feature Results	44
5.6	Improvement Suggestions	44
5.7	Threats to Validity	45
5.7.1	Test Subjects	45
5.7.2	Introduction to Controls	45
5.8	Conclusion	46
6	Related Work	47
7	Conclusions and Future Work	49
7.1	Conclusions	49
7.2	Future Work	49
	Bibliography	51
	Appendix A	
	Appendix B	
	Appendix C	

Introduction

1.1 Motivation

In 2015, a virtual reality (VR) mode was introduced to ExplorViz¹ for the first time [Krause 2015]. It was migrated to the new microservice architecture and further developed by [Häsemeyer 2017] two years later. Since then large software landscapes can not only be experienced on a monitor, but also in a virtual environment. Software development is usually done in teams. If one wants to use ExplorViz collaboratively today, one has to sit down together in front of a screen to experience the same thing. However, only one person can interact with the software at a time. This approach is particularly difficult when developers want to use ExplorViz beyond national borders. To make ExplorViz more collaborative and to actively involve all team members, a collaborative mode should be added.

In this bachelor's thesis the VR extension of ExplorViz is extended by a collaborative mode. This shall enable users to experience software landscapes together in an immersive way. They can work with one another regardless of location. They can meet each other in the virtual world and see movements and actions of others in real time. Users can work simultaneously to manipulate software landscapes and application models.

1.2 Goals

In this sections the goals of our bachelor's thesis are presented.

1.2.1 G1: Concept/Design

Before we start with the implementation, we develop a concept for the multi-user mode. We plan which components and services are added to the existing ExplorViz VR extension. We also plan which protocols are used to exchange data, such as user's position data. The additional effort, which results from the conception, allows a more systematic implementation and distribution of the tasks to the team. The goal is to develop a concept that enables the collaborative and geographically independent use of ExplorViz VR and extends it with useful features.

¹<https://www.explorviz.net>

1. Introduction

1.2.2 G2: Implementation

ExplorViz is extended by the collaborative VR mode worked out in the design. A backend extension is required for this. This extension stores data about the users and the landscape and distributes it to other users. The backend extension is responsible for synchronizing the data of the users. Changes are also made in the frontend VR extension. It must send the necessary data to the backend extension, but it must also process the incoming data and display it correctly.

1.2.3 G3: Evaluation

Once the multi-user mode has been implemented, its added value is to be verified. The aim is to evaluate whether further development would be useful and how this mode could be improved in the future. Furthermore, the added value and usability of individual features are to be evaluated. To this end, test subjects will test the mode and evaluate it on the basis of a developed questionnaire.

1.3 Document Structure

Chapter 2 explains the fundamentals of the technologies and tools used in this work. Based on that, Chapter 3 presents the concept and design of a new backend extension and the extension of the existing frontend extension. This is followed by the implementation of the design in Chapter 4. How well the implementation is received is examined in the study and discussed in Chapter 5. An analysis of related work and comparison with ours is given in Chapter 6. In Chapter 7 we summarize the work and give an outlook on how the new mode may be improved in the future.

This bachelor's thesis is the result of a bachelor's project in collaboration with Malte Hansen. His thesis describes the development from the HTC Vive's point of view [Hansen 2018].

Foundations and Technologies

This chapter explains the fundamentals and technologies relevant to the bachelor's thesis.

2.1 Virtual Reality

The idea of VR has been around for centuries. In 1968, Ivan Sutherland and Bob Sproull developed the first VR head-mounted display (HMD) [Sutherland 1968]. It used a stereoscopic display and the image seen on it was updated depending on the user's head position and rotation. However, the device was so heavy, it had to be carried by the ceiling. Today's VR headsets weigh much less, such as the Oculus Rift¹ and Oculus Go² both with less than 500 grams.

The Oculus Rift is a head-mounted display released in March 2016 as a consumer version. It has a resolution of 2160x1200 pixels (1080x1200 per eye), a refresh rate of 90Hz and built-in headphones [Farahani et al. 2016]. To use it, two Oculus Sensors are needed to track the position of HMD and controllers. The controllers launched by Oculus VR are called Oculus Touch and offer greater possibilities for interaction in the virtual world through movement, but also through buttons and analog sticks. Both the Oculus Rift and the Oculus Touch [Anthes et al. 2016] are equipped with infrared LEDs that enable tracking by the sensors. The complete set, as used during our development, can be seen in Figure 2.1. For the evaluation, we used an additional sensor to enable room-scale tracking.

¹<https://www.oculus.com/rift/>

²<https://www.oculus.com/go/>

2. Foundations and Technologies



Figure 2.1. Oculus Rift set consisting of Oculus Rift, 2x Oculus Touch and 2x Oculus Sensor³

2.2 ExplorViz

ExplorViz is a web application that can monitor and visualize large software landscapes at runtime [Fittkau et al. 2013]. In the visualization, systems, modes, applications, packages and classes are displayed in landscape and application views [Fittkau et al. 2015b]. Communication collected through monitoring is also visualized.

Figure 2.2 shows the landscape view of ExplorViz. Gray boxes represent systems in a system landscape. The green boxes in the systems are called nodes. A node can contain different applications. Communication between applications is represented by orange lines. The thicker the line, the more communication takes place. In order to keep the view clear, individual systems and nodes can be folded in and out as required [Fittkau et al. 2017].

For further details, you can view individual applications in the application view (see Figure 2.3). The green boxes represent the packages or package hierarchy. These can be opened and closed as desired. At the top level there are packages that contain classes. Classes are displayed as purple boxes. The height of these is determined by the number of instances. The higher the box is, the more instances there are [Krause 2015]. The communication between classes is represented by orange lines as in the landscape view.

³<http://gizbrain.com/wp-content/uploads/2016/01/Oculus-Rift-VR-set.jpg> gizbrain.com/wp-content/uploads/2016/01/Oculus-Rift-VR-set.jpg gizbrain.com/wp-content/uploads/2016/01/Oculus-Rift-VR-set.jpg

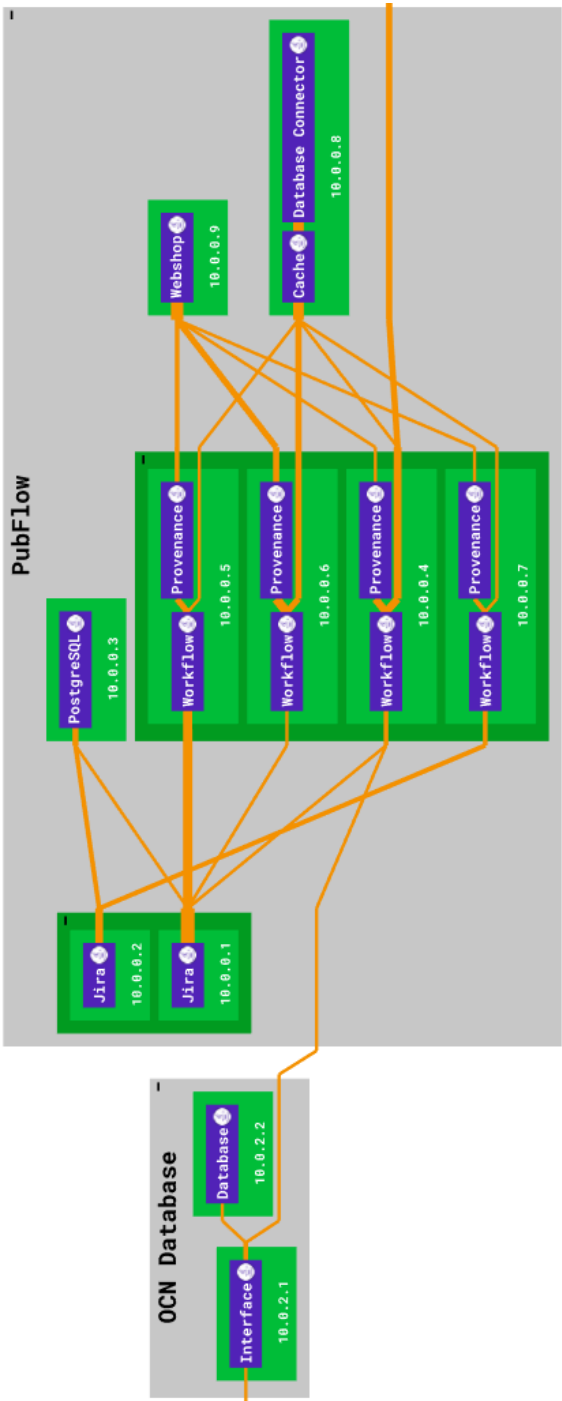


Figure 2.2. Illustration of the landscape view of ExplorViz

2. Foundations and Technologies

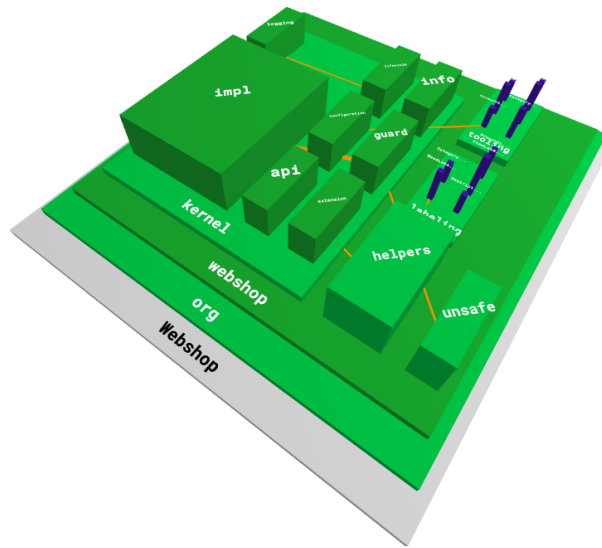


Figure 2.3. Illustration of the application view of ExplorViz

If you want to view the model of an application in the real world, you can export a 3D model in ExplorViz for 3D printing [Fittkau et al. 2015a]. A sample 3D application print is seen in Figure 2.4.

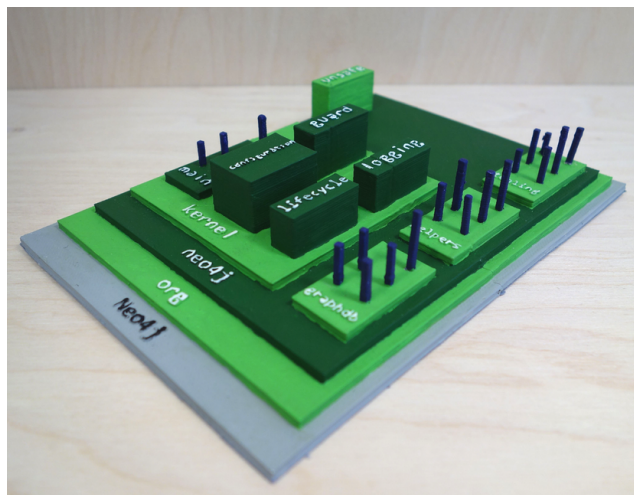


Figure 2.4. 3D printing of a 3D visualization with ExplorViz⁴

2.3 Three.js

Three.js⁵ is a JavaScript (JS) library that allows the creation, rendering and display of 2D and 3D objects in the web browser. At the beginning a Scene is needed, in which objects can be placed. In addition, a Camera is needed, which indicates the position and the field of view from which the scene will be rendered. There are different types of cameras. Perspective cameras imitate the view of a human being, for example. Rendering is done with the WebGLRenderer⁶. It creates the images of a scene, which are displayed in the web browser on the so-called Canvas. The continuous refreshing is usually solved with a recursively calling loop.

The most important classes in Three.js are described in the following.

▷ THREE.Scene⁷:

A THREE.Scene can be understood as a three-dimensional room where objects (THREE.Object3D), lights and cameras can be placed. The THREE.Scene itself inherits from THREE.Object3D.

▷ THREE.Mesh⁸:

Represents a triangle mesh. It consists of a THREE.Geometry and a THREE.Material. A mesh is a collection of vertices, edges and faces, which are stored a Geometry. A vertex is a single point. An edge connects two vertices with a straight line. In a triangle mesh, a face is the area or surface in between 3 edges that shape a triangle. The material describes the look of the THREE.Mesh. Meshes can be added to a THREE.Scene, since the class inherits from THREE.Object3D.

▷ THREE.Geometry⁹:

A THREE.Geometry describes a shape using things like vertices and faces. There are multiple predefined geometries describing shapes such as boxes (THREE.BoxGeometry¹⁰), spheres (THREE.SphereGeometry¹¹) or a flat shape (THREE.PlaneGeometry¹²). Latter is like a box with zero height.

▷ THREE.Material¹³:

Describes the appearance of an object. There are multiple predefined materials. The most basic material in Three.js is called THREE.MeshBasicMaterial¹⁴. It is not affected by

⁴https://www.explorviz.net/features_phys

⁵<https://threejs.org>

⁶<https://threejs.org/docs/#api/en/renderers/WebGLRenderer>

⁷<https://threejs.org/docs/#api/en/scenes/Scene>

⁸<https://threejs.org/docs/#api/en/objects/Mesh>

⁹<https://threejs.org/docs/#api/en/core/Geometry>

¹⁰<https://threejs.org/docs/#api/en/geometries/BoxGeometry>

¹¹<https://threejs.org/docs/#api/en/geometries/SphereGeometry>

¹²<https://threejs.org/docs/#api/en/geometries/PlaneGeometry>

¹³<https://threejs.org/docs/#api/en/materials/Material>

¹⁴<https://threejs.org/docs/#api/en/materials/MeshBasicMaterial>

2. Foundations and Technologies

light. Its brightness can be set and stays the same whether it's hit by light or is placed in a shadow. To use a texture as material, the `.map` property can be set.

▷ `THREE.Object3D`¹⁵:

Is the base class for most objects. `THREE.Object3D`'s have multiple properties such as position, rotation and scale. They can also be used for grouping. A `THREE.Object3D` can have at most one parent, but have multiple children. This enables a tree-like object structure. Moving a parent will also move its children. However, it will not change the childrens' local position, for the position relative to the parent did not change. Same goes for rotation and similar. For clearer grouping on a syntactic level, the class `THREE.Group`¹⁶ is used. It inherits from `THREE.Object3D` and is almost identical to it. `THREE.Object3D` can be added to the `THREE.Scene`.

Using the JS WebVR¹⁷ API, `Three.js` can also be used for virtual reality. VR devices can be detected, their position or orientation can be retrieved and scenes can be displayed.

`ExplorViz` uses `Three.js` and `WebVR` for VR visualization. Therefore they have an important role in collaborative mode.

2.4 ExplorViz VR Extension

The VR mode in `ExplorViz` today was introduced by Häsemeyer in 2017 [Häsemeyer 2017]. His extension allows the single-user exploration of software landscapes using HMDs such as the Oculus Rift or HTC Vive. In the following, we refer to his version as VR1.1¹⁸.

2.4.1 Application and Landscape View

Both, the application and landscape perspective, can be experienced in VR. While these two views are else kept separate in `ExplorViz`, the views are fused in this VR mode. On the floor one finds the landscape view as a 3D model. The systems are displayed as three-dimensional gray boxes with their respective name labeled on top. When opened, they contain green nodes, which beside the name on top also have their running applications displayed on them. When an application is opened, it appears on top of the landscape and can be moved and placed to the preferred position. Different to the browser view, both the landscape and one application can be seen and explored simultaneously.

2.4.2 Interaction

The user can move through the virtual world by physically moving in the real world. They can move their head to look around or crouch to get a closer look at the landscape. The

¹⁵<https://threejs.org/docs/#api/en/core/Object3D>

¹⁶<https://threejs.org/docs/#api/en/objects/Group>

¹⁷<https://webvr.info/>

¹⁸<https://github.com/ExplorViz/explorviz-frontent-extension-vr/tree/v1.1>

2.4. ExplorViz VR Extension

tracking enabled by the base stations of the HTC Vive or the sensors of the Oculus Rift allow this movements, since the HMD's position and rotation is known. To use ExplorViz VR to its full extend, two VR controllers, one for the left and one for the right hand, are needed. In the virtual world these controllers are also displayed. However, the only model available is the HTC Vive controller's model. A user with Oculus Touch controllers will thus also see HTC Vive controllers. Attached to both controller models is a "laser ray" or line, that comes out of the controller. It is used to point at objects and to interact with them. In the following, we explain what the user can do and what they can interact with in the virtual world. We also explain the controls. However, the reader is expected to know about the button layout on either the Oculus Rift or HTC Vive for now.

Pointing the left controller at the floor displays a blue circle where it is hit. Pressing the trigger button on the left controller will teleport the user to the blue circle's position. This feature allows faster movement.

Pointing the right controller at a system and pressing the trigger on the right controller unfolds it and the contained nodes show up. Clicking at an application in the same way, opens the corresponding application model. Components in the application model are opened in the same way as systems. Open systems or components can be closed in the same way that they are opened.

Components and classes in an application are highlighted by pointing the left controller at one and pressing the trigger on it. If a component or class is highlighted, it is displayed in red. Using the highlighting on it again, unhighlights it.

To view more details on classes, components, systems, nodes or applications the user can point either controller at one and press the respective grip button. This opens a text box next to the controller with more details on it, e.g., contained classes in a component or the number of instances of a class. This text box is closed by pointing at the sky and pressing the button again.

An application can be moved. Pointing either the left or right controller at the application and pressing and holding down the analog stick or touchpad on it will bind the application to the controller. As long as the application is attached, it is moved by physically moving the controller. When the button is released, the application is unbound.

Concept/Design

This chapter introduces the general idea of a multi-user mode and provides a concept for it.

3.1 General Concept

VR1.1 is conceptualized for one-person usage. Since software is often developed in a team, working with two or more persons may be useful. A multi-user mode allows multiple users to come and explore together in one virtual room. A concept drawing of such a mode is displayed in Figure 3.1. The world, i.e. landscape, applications, are synchronized. That means, if one user makes a change to e.g. the landscape, it is updated for other users as well. This is the only way that users can work together successfully and develop the feeling of being in the same virtual room. Both landscape and applications can be manipulated by any user. I.e., anyone can open components, highlight them and move applications. Highlighting can be seen by any users and the color depends on which user highlighted the component.

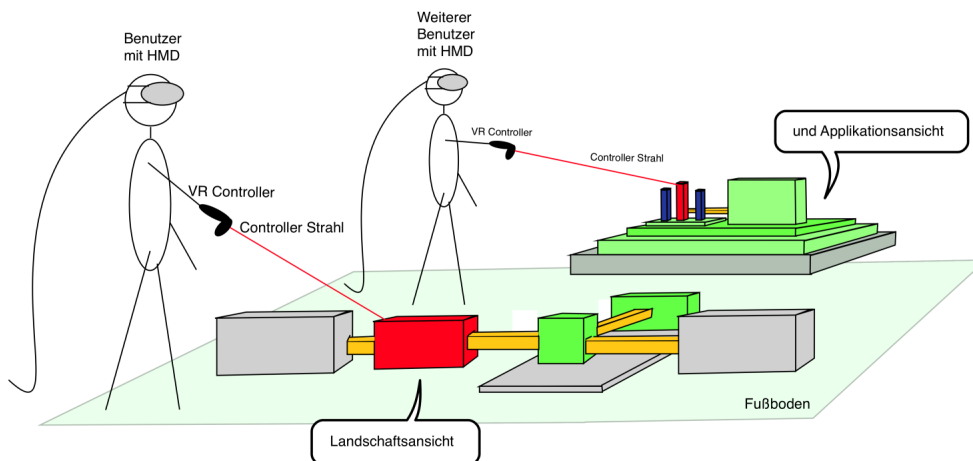


Figure 3.1. Concept drawing of collaborative software exploration in Explorviz [Häsemeyer 2017]

3. Concept/Design

3.2 Architecture

ExplorViz uses a microservice architecture as displayed in Figure 3.2. Frontend and backend of ExplorViz are two distinct microservices. These can be deployed on two different servers. The backend provides a HTTP based RESTful API for communication with the frontend. Frontend and backend can both be enhanced with extensions. For this purpose, ExplorViz provides dummy extensions for both backend and frontend as a starting point. They can read data from the respective core via a defined API. [Zirkelbach et al. 2018]

ExplorViz' backend is written in Java and uses Gradle¹ as build tool. The frontend is however written in JS and uses the Ember.js² (Ember) framework.

The data exchange between frontend extension and backend extension usually also takes place via a RESTful API. Data is transferred via HTTP, which is a layer built on top of the TCP layer. Therefore, it adds some overhead, which we would like to avoid for the multi-user approach. We therefore choose the WebSocket protocol. It is a bi-directional protocol, which enables both the client and server to send messages. Furthermore, client and server both communicate over the same TCP connection. While WebSockets are not pure TCP, they are more lightweight than HTTP and a WebSocket connection is ten times more efficient than a HTTP connection [Qigang and Sun 2012].

ExplorViz features a star topology between frontend and client. The frontend is the central point to which all clients are connected. In particular, the clients are not connected amongst each other. We use this topology for the multi-user approach as well. We will have a backend extension based on a backend dummy extension. For the frontend side, we will further develop the existing VR1.1. The connection between them will be a start topology with the backend extension as center point (see Figure 3.3).

For data exchange, we choose the JavaScript Object Notation (JSON) data-interchange format, because it is lightweight and commonly used in nowadays web applications.

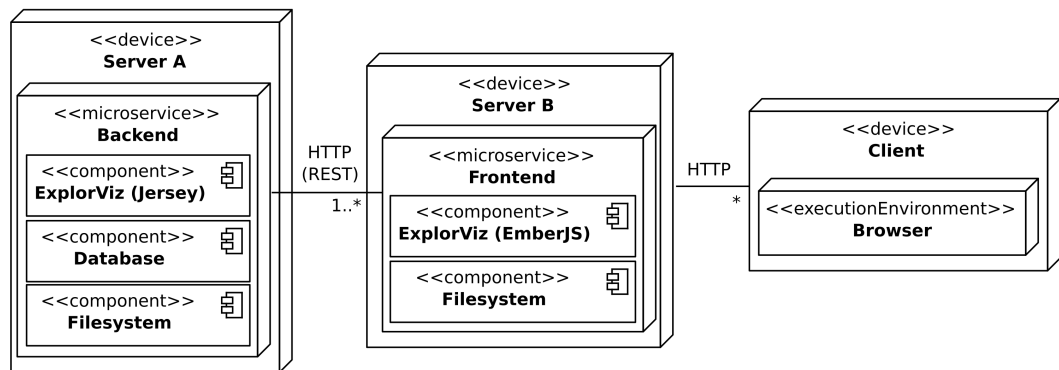


Figure 3.2. The new microservice architecture of ExplorViz [Zirkelbach et al. 2018]

¹<https://gradle.org/>

²<https://www.emberjs.com/>

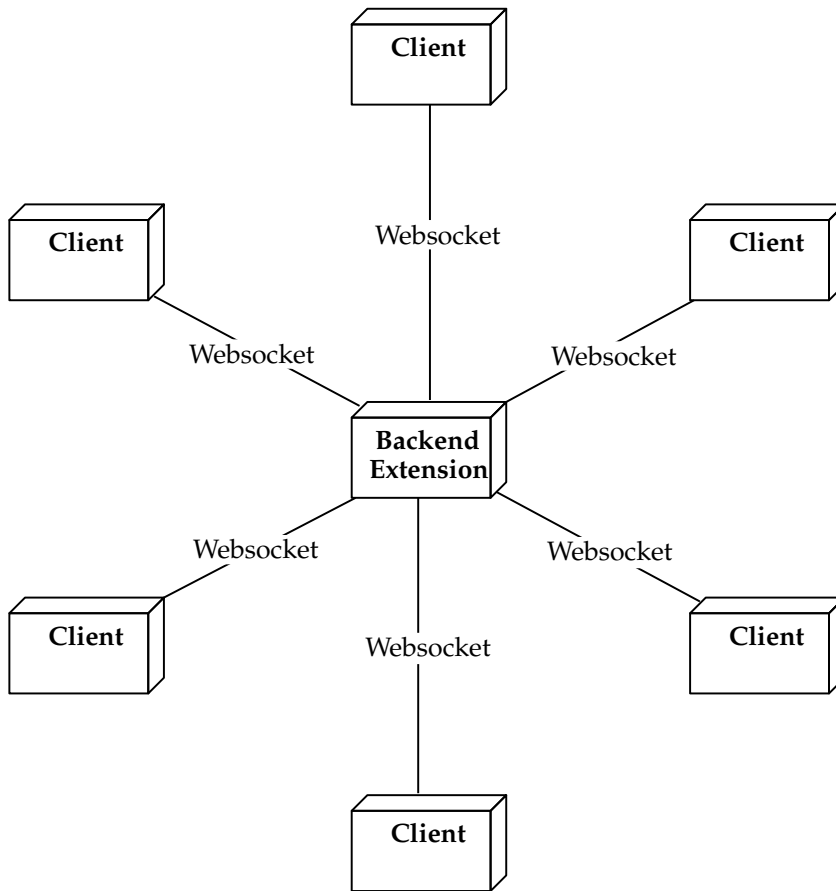


Figure 3.3. Star topology between backend extension and frontend extension (client)

3.3 Frontend

Since a VR frontend extension for ExplorViz (VR1.1) already exists, we will build on it and develop it further.

3.3.1 Landscape and Applications

Working alone in VR1.1, a user can do multiple things such as opening systems, applications and the included components. Applications can be moved and components be highlighted. All those actions should be available in multi-user mode as well. However, when a user performs actions modifying either the landscape or application, other users' landscape, respective application, has to change accordingly. Therefore, the frontend extension should

3. Concept/Design

send out messages to the backend extension every time such actions are performed, which then distributes the changes to the other clients. Furthermore, the frontend extension also needs to be able to process data coming in regarding those changes.

In VR1.1, only one application can be open at a time. While this might seem reasonable for a single user, this could be a potential bottle neck in multi-user mode. Users could not work in parallel on different applications. Therefore, we want to allow multiple applications at a time to enhance efficiency.

One feature, if so little, is the highlighting of components inside applications. Currently, the color the component is highlighted in is always red. Since this may be confusing if multiple users highlight something, we want every user to be assigned a unique color. This assignment should however be handled by the backend extension.

3.3.2 User Visualization

Users shall be able to see each other inside the virtual room. However, not their physical appearances or faces, but rather an abstract human representation or avatar. In Figure 3.1, the users are visualized as stick persons with an HMD attached to their heads and holding a controller. Showing the controllers including the laser rays can be useful. That way the other users know where one's hands are and where one points at. Animating a body, even if it is just a simple one, may be difficult. Especially since the only data that is available is the rotation and position of the HMD and the controllers. Therefore, we decide to only show the controllers and HMD, but not the body.

3.3.3 Options Menu

The number of keys on the controllers are limited. Therefore, new features should be bundled inside a menu (see Figure 3.4). The menu shall be opened by pressing a key on one of the controllers and be attached to the left controller. The right controller with its laser ray can then be used to point at buttons and be used to interact with it in general. In the following, the features, as seen in Figure 3.4, are further explained.

Change Camera

It can happen that the height of the camera in the VR application is not right due to a bad room setup. Adjusting the height in the application itself is faster than a new setup and allows for a temporary fix. Furthermore, the feature allows to work closer to the landscape and read labels better when setting the camera height lower.

Move Landscape

In VR1.1 the landscape is always positioned in the center of the floor. We suspect, that longer work sessions in ExplorViz VR may cause neck pain, since a user has to constantly

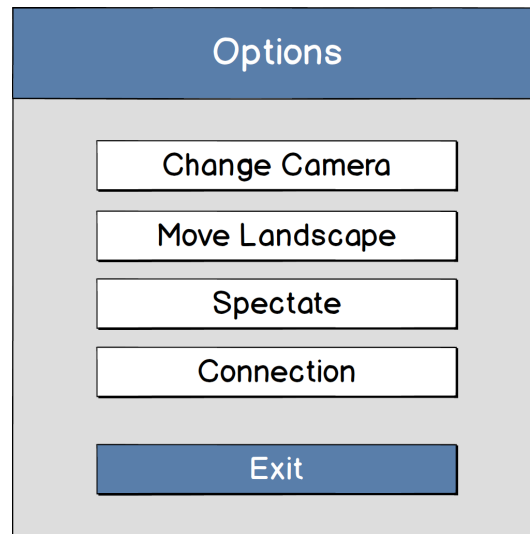


Figure 3.4. Mockup of the concepted options menu

look down on the floor to work with the landscape. Therefore, we want to add a menu in which the landscape can be rotated, moved and changed in height. This would allow positioning the landscape as preferred, e.g., directly in front of the user as if it were placed on a wall.

Spectate

The multi-user mode enables exploration of software landscapes with multiple users. However, when multiple users stand in front of, e.g., an application, they might stand in each other's way. Additionally, some users might prefer to just watch another user do and show something. Therefore, we want to add a spectate feature which will allow users to slip into another user's perspective.

Connection

ExplorViz VR shall be usable in single and multi-user mode. That way users can decide whether to work alone or in a team. The user shall be able to connect and disconnect through a menu. This menu shall also display the current connection state. That way, the user can always find out whether they work online or offline.

3. Concept/Design

3.3.4 User List

The collaborative mode is expected to be used in a team. Furthermore, users are expected to be communicating during use. They could be in the same physical room for this or using VoIP. Even though we expect users to usually know who they are with in the virtual world of ExplorViz, we still want to add a user list as depicted in Figure 3.5. That way users can at any time get an over of the users currently connected. Furthermore, the user can see what users are spectating and are thus not visible in the virtual room.

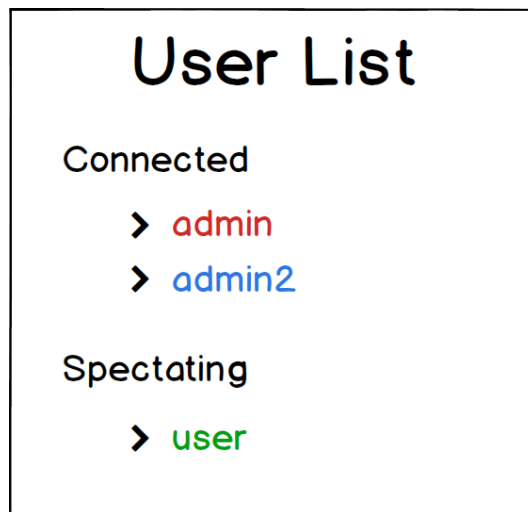


Figure 3.5. A concept drawing of the user list

3.3.5 Overlays

There shall be an option to inform a user about more or less important events. Therefore, we decide to add two different kinds of UI elements.

Notifications

We call notifications a type of message display for low-priority messages. It is a small window (see Figure 3.6) displayed at the top of the display for a short period of time (two to three seconds). Its position and size should make it readable and slightly noticeable, but not distracting or even blocking the view. It is used to display events such as

- ▷ a user (dis-) connects
- ▷ a user enters/leaves spectate mode

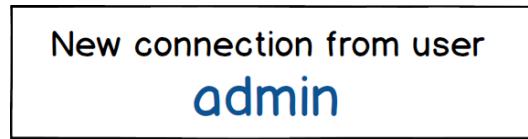


Figure 3.6. Mockup of the notifications display

Hints

Different to notifications, hints (see Figure 3.7) are used to display important messages. Since they are of great priority, they are displayed in the middle of the screen. That way, the user can easily notice them. Such hints could be

- ▷ missing details when trying to open an application
- ▷ lost connection
- ▷ application is already being moved by another user

This shall improve usability and help the user understand why, e.g., an application can not be opened.



Figure 3.7. Mockup of the concepted hints display

3.4 Backend

The backend is the central point of the star topology. However, in order to avoid having to change the actual backend, ExplorViz allows for the creation of backend extensions, similarly to the creation of frontend extension. This is also supported in the backend. Based on a dummy backend extension³, the VR backend extension is created. The backend extension shall run a WebSocket server. That way, a connection can be established by the frontend extension. It shall store a list of the connected users including their positions, names, colors, but also if they are spectating or not. Beside user data, the backend extension also keeps the current state of the landscape and the applications. In addition to storing all necessary data and processing sent data from connected clients, one of the key tasks is to distribute it. E.g., if a client sends the position data of their controllers, the backend extension needs to inform all other users about that change. Therefore, every time a client

³<https://github.com/ExplorViz/explorviz-backend-extension-dummy>

3. Concept/Design

sends data, the backend extension has to decide if other clients need to be informed and send updates accordingly. The processing of messages should be carried out in parallel using threads. This requires necessary synchronization measures during development. However, we expect this to result in faster processing of the data sent by the clients.

Implementation

4.1 Frontend

In this section we explain how our frontend concept is implemented. The full implementation is found on GitHub¹.

4.1.1 WebSockets

As explained in Chapter 3, we use WebSockets to interchange data between clients and server. The client is the ExplorViz frontend extension and the server the ExplorViz backend extension. Since our frontend runs Ember, we add the `ember-websockets`² addon. This allows us to include WebSockets as a service.

Establish Connection

The WebSocket connection between frontend extension and backend extension is not automatically established on page load. The user rather gets a choice, whether they want to explore alone in single-user or collaboratively in multi-user mode. How to connect or disconnect when in VR is explained in Section 4.1.4.

The address and port the socket will connect to is defined in a separate configuration file `config_multiuser.json`. The address must match the server's ip address and the port the one that is configured in the backend extension. When the user presses the connect button, a WebSocket connection is opened on the port and address that are set (see Listing 4.1, line 1). Directly afterward, the needed handler functions are set. The open handler, as seen in line 2, is the function that is called when the connecting was successful. The message handler (line 3) is called every time a message from the server is received. The corresponding function decided what to do with the received data. The close handler (line 4) is called when the connection is interrupted or the server closes the socket.

If the connection is successfully established, the server sends the client a message with the event `receive_self_connecting`, including the user's id and color. Since the server does not know the user's name yet, the frontend extension sends back a message with event `receive_connect_request` including the name. This name matches the user's authentication

¹<https://github.com/ExplorViz/explorviz-frontend-extension-vr/tree/v2.0>

²<https://github.com/thoov/ember-websockets>

4. Implementation

```
1  const socket = this.get('websockets').socketFor('ws://${host}:${port}/')
2  socket.on('open', this.openHandler, this)
3  socket.on('message', this.messageHandler, this)
4  socket.on('close', this.closeHandler, this)
```

Listing 4.1. Setting up and establishing the websocket connection

```
1  if (emberModelName === "system") {
2    this.trigger('systemStateChanged', emberModel.id, emberModel.get('opened'))
3  } else if (emberModelName === "nodegroup") {
4    this.trigger('nodegroupStateChanged', emberModel.id, emberModel.get('opened'))
5  }
```

Listing 4.2. Triggering of events when the landscape is manipulated

name. The server then registers the user as connected and sends all data necessary, e.g., other users' names, ids, states and current landscape.

4.1.2 Landscape and Applications

Landscape

To keep the landscape synchronized, the frontend extension has to inform the backend extension about any changes. There are, however, only a few actions that can be performed. I.e. systems and nodegroups can be opened or closed. When a system or nodegroup is selected for opening, an event called `systemStateChanged`, respectively `nodegroupStateChanged`, is triggered. This can be seen in the code snippet in Listing 4.2. Additionally, the unique identifier of the system, respectively nodegroup, is added and its current state.

In VR1.1 these events are caught and used to update the landscape. In our implementation we additionally catch those events inside the multi-user component and send a message with the event `receive_system_update` or `receive_nodegroup_update` to the backend extension. It contains both the id and a boolean describing whether the system/nodegroup was opened or closed. The backend distributes this data to the other users.

When the data is received by the client, the system or nodegroup with the corresponding id is updated. In Listing 4.3 the updating process for systems is listed. Every time a system is closed or opened by another user, it is necessary to iterate over all systems in the landscape in the worst case (line 1). For every of the systems it is checked if it is the system matching the id (line 2). The system that matches the id is set to either opened or closed (line 3). At the end the landscape is redrawn (line 4). The `ExplorViz` frontend uses a `Store` to store applications. However, sometimes applications are not in the store and thus can not be accessed although they exist. Therefore, the iteration over all the systems is necessary

```

1  this.get('vrLandscape').children.forEach(function (system) {
2    if (system.userData.model && system.userData.model.id === id) {
3      system.userData.model.setOpened(isOpen)
4      self.populateScene()
5      return

```

Listing 4.3. Update of the landscape when a system is changed

```

1  let highlightObj = {
2    "event": "receive_highlight_update",
3    "appID": appID,
4    "entityID": entityID,
5    "isHighlighted": isHighlighted,
6    "color": color
7  }

```

Listing 4.4. Data sent when an a class or component is highlighted

for now.

Applications

In VR1.1 it was not possible to have open more than one application at a time. As described in the concept, we want to change that. Instead of storing just one application and its corresponding id, we use a Map that we call openApps. It maps an application's id to the corresponding object. That way, when an application is manipulated, the id can be used as a reference.

The id is used to, e.g., update highlighting. When a user highlights or unhighlights a component or class, the backend extension is informed. Therefore, the frontend extension sends a message as seen in Listing 4.4. It contains the application's id (line 3), the id of the component or class (line 4) and whether the component or class is now highlighted or not (line 5). There is, however, also a key called color included (line 6). This is not the color of the highlighting. It is rather the original color of the component or class. We ran into the issue that unhighlighting would not restore the color correctly. The component was brighter after once highlighting it. We avoid this issue by including the original color, which is then used by the other clients to restore the color properly. How highlighting looks in VR, is shown in Figure 4.1.

4. Implementation

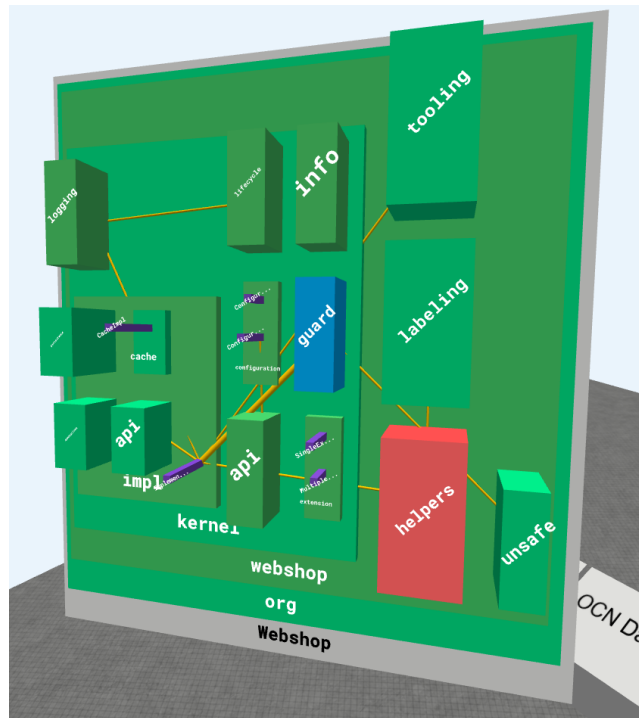


Figure 4.1. Application with two components highlighted by different users

Another part of our implementation regarding applications is devoted to movement. When one user grabs onto an applications, not only should the position and rotation of the application be updated for any other user, but other users shall be blocked from grabbing it. Therefore, the frontend extension sends out a message containing the application id and its position and rotation. Sending position and rotation makes sure, the other clients know in exactly what condition the application was when moving began. Furthermore, the message contains with which of the both controllers the app is grabbed and additionally the controller's position and rotation. Using the position and rotation, the controller can as well be put into position. The controller is a `THREE.Mesh`. It can thus have child objects. The application is added to the controller as a child. That way, when the controller's position or rotation is updated, the application moves with it. This way, the frontend extension does not need to send out the position and rotation of the application itself. The controller position, which is sent out anyway, can be reused. When this data is received, the frontend extension of those users, which did not grab onto the application, adds the application's id to a Set stored inside a variable named `boundApps`. When the user tries to grab an application that is already being moved, the user can not, since it is contained in `boundApps`. The user, furthermore, gets a notification.

```

1  teleportToPosition(position) {
2      const cameraOffset = new THREE.Vector3()
3      cameraOffset.copy(this.camera.position)
4      cameraOffset.y = 0
5      this.user.position.subVectors(new THREE.Vector3(position.x, this.user.position
        .y, position.z), cameraOffset)

```

Listing 4.5. Teleporting the user by moving the user group

4.1.3 Teleportation

In VR1.1 the user teleportation is realized not by moving the camera position and the controllers' positions, but by moving the the world. I.e. the floor, landscape and application. While this does not raise problems in single-user mode, it does in multi-user mode. The user's position would need to be calculated as the sum of camera position and the world's position offset. We decided to take a different approach.

Our first approach was to stop moving the world, but instead move the user's camera. However, we found that the position is constantly overwritten by the HMD's position data. So this was not longer an option.

Our second approach is to group camera and controllers in a `THREE.Group`. That way, the user can be moved as a whole. Furthermore, when the camera's position is overwritten by the HMD's position, it only changes the local position inside the group. The same goes for the controllers. The group itself can still be moved, moving both camera and controllers with it.

When a user uses the teleport feature, the function `teleportToPosition`, as seen in Listing 4.5, is called with the position on the floor the user pointed at. In line 2 and 3 a copy of the current camera position is created. This is the local position inside the group, not a world position. It describes how far off the camera position is from the center point of the group. That is why it is called `cameraOffset`. Then in line 5, the group's position is set to the difference between the new position and the camera's offset inside the group. That way, the group's new position is set to the exact value, such that the camera's new position matches the value in the position's value. However, the user's height is kept (y position). Teleporting would else position the user inside the floor.

4.1.4 Menus

Menus are our idea of displaying something on a plane geometry and making it interactable. We created an Ember `Util` called `Menu`. That way, new `Menu` objects can be created.

Menus have multiple properties, three of them can be seen in Listing 4.6. In line 1 we see the `name` property. It can be set and should be a unique identifier. The `THREE.Mesh`, which the menu will be added as to the `Scene`, will get this identifier as well. That way the `Mesh`

4. Implementation

```
1  name: null,  
2  size: { height: 0.3, width: 0.3},  
3  resolution: { width: 512, height: 512 }
```

Listing 4.6. A selection of menu properties

can be found in the Scene and later on be deleted, when the menu is closed. The Geometry of the Mesh will have a specified size, which matches the size property in line 2. The size has a default value, can however be set to what is preferred. The Texture of the Mesh will be created by drawing the menu elements on a HTML Canvas, that is then converted to a Texture. The resolution of the Texture matches the value of the resolution property in line 3.

The menu can be filled with elements such as texts, text buttons or arrow buttons. Texts are used for display and, e.g., can be used to add a title to a menu. A text button is like a text but it can be interacted with. It has a rectangle shaped background to make clear, it can be clicked. Arrow buttons can also be interacted with, however, they come in different shapes.

Interaction

Menus can be used to display data or to have elements in them to interact with. In VR1.1, when the right trigger is pressed, a function named `onTriggerDownController2` in `interaction.js` is called. This function checks what object the user pointed the right controller at and, e.g., if it is a system in the landscape, it is opened. As seen in Listing 4.7, the code now checks for menus. In line 5, it is tried to retrieve an opened menu that matches the name of the intersected object. If such a menu exists, its `interact` function is called with the action and position the laser ray intersects at (line 7). Every menu defines its own `interact` function.

Options Menu

Change Camera

The menu used to change the camera height, or more precisely the user's height, is depicted in Figure 4.2 (❶). The menu displays the current height in the center of the menu. The buttons to the left and right are used to decrease or increase the user's height, respectively. Listing 4.8 shows the implementation of this. Depending on which button is pressed (line 1 / line 4), the position of the user group (`THREE.Group`) is either lowered or raised (line 2 / line 5). The text in the center showing the current height is then updated to show the new height (line 3 / 6).


```

1  onTriggerDownController2(event, objects) {
2    ...
3    const intersectedViewObj = this.get('raycaster').raycasting(origin, direction,
      null, objects)
4    ...
5    let menu = Menus.get(intersectedViewObj.object.name)
6    if(menu) {
7      menu.interact('rightTrigger', intersectedViewObj.uv)
8      return
9    }
10   ...
11  }

```

Listing 4.7. Handling right trigger interaction with menu

```

1  if(item.name === 'height_down') {
2    this.get('user').position.y -= 0.05
3    menu.updateText('camera_height', this.get('user').position.y.toFixed(2))
4  } else if(item.name === 'height_up') {
5    this.get('user').position.y += 0.05
6    menu.updateText('camera_height', this.get('user').position.y.toFixed(2))
7  }

```

Listing 4.8. Changing the user's hight via menu

Move Landscape

Our implementation allows the users to move the landscape to a position they prefer. In Figure 4.2 (●) the menu is displayed. It features multiple buttons. In the upper left are two buttons. They are used to move the landscape up or down. In the middle there are four buttons, which allow to move the landscape along the floor. Lastly, the menu features 2 buttons in the upper right, which are displayed as rotating arrows. They allow rotating the landscape.

In VR1.1, the landscape is re-centered every time the landscape is manipulated. Re-centering means that the center of the landscape is aligned with the center of the floor. So, if the landscape were moved, it would always go back to the center of the floor every time the landscape changes.

Our first approach to avoid this was to remove the re-centering feature completely, however the landscape was always at different positions depending on which systems where opened. So we took another aproach. In this we kept the re-centering feature, but added an `environmentOffset`. This offset stores how far away the user moved the landscape from the center point of the floor. Every time the landscape is re-centered, the offset is then

4. Implementation



Figure 4.2. The 4 submenus inside the options menu

added to the landscapes position again. This moves it to the right position again.

In Listing 4.9 the `moveLandscape` function is displayed. When using the menu for movement, it is called with a fixed `delta` value, which tells the function how far the landscape should be moved from its current position. In line 2-4 the `environmentOffset` is adjusted. After that an event called `landscapeMoved` is triggered (line 7). It is caught in a different part of the extension and results in the change being sent out to the backend extension. That way, other users' landscapes will be adjusted.

Spectate

The spectate menu seen in Figure 4.2 (③) allows the user to select another user they want to watch. They slip into their view. Their position is locked to the position of the user

```

1  moveLandscape(delta) {
2    this.get('environmentOffset').x += delta.x
3    this.get('environmentOffset').y += delta.y
4    this.get('environmentOffset').z += delta.z
5    ...
6    let deltaPosition = new THREE.Vector3(delta.x, delta.y, delta.z)
7    this.get('interaction').trigger('landscapeMoved', deltaPosition)
8  }

```

Listing 4.9. Move landscape by changing its offset

```

1  let position = spectatedUser.get('camera.position')
2  const cameraOffset = new THREE.Vector3()
3  cameraOffset.copy(this.get('camera.position'))
4  this.get('user.position').subVectors(new THREE.Vector3(position.x, position.y,
    position.z), cameraOffset)

```

Listing 4.10. Update camera position when spectating

they spectate. However, they can still look around freely. How the view looks from the spectator's perspective is seen in Figure 4.3. The spectator still sees their own controllers and the spectate menu. However, they can no longer interact with the landscape or applications. Spectators are hidden to others. Their HMDs, controllers and name tags are set invisible for others.

To achieve the user position being locked to the spectatee's position, the user's position has to be explicitly set on every frame. How that is achieved is shown in Listing 4.10. The frontend extensions always has the position of other user's HMDs, since it is necessary for visualizing them. In line 2 we retrieve the spectatee's HMD (camera) position. Then in line 3-4 a copy of the user's own camera position is created. We introduced a user group earlier, which contains camera and controllers. To move the user to the spectatee's position, it has to be calculated how far off user's camera is from the spectatee's camera and the user group be set accordingly. This is what happens in line 5. If done every frame, the spectator's camera will always be at the camera position of the user they watch.

Connection

We leave it up to the user to decide whether they want to work in single-user mode or collaboratively. Therefore, we created a menu regarding the connection to the backend extension. It is seen in Figure 4.2 (4) and contains the current status (offline, connecting, online) plus a connect/disconnect button. How the connection is established is presented in Section 4.1.1.

4. Implementation



Figure 4.3. User using the spectate feature

4.1.5 User Visualization

HMD and Controllers

As explained in the concept, users should be able to see each other in the virtual environment. Instead of a human-like model or an abstract stick person (see Figure 3.1), we focus on a different approach. As seen in Figure 4.4, we introduce only an HMD model and the controller models for the user model. That means a user does not see any bodies, but the devices used by the other users. The HMD model does not change depending on which device is used. It is rather a generic model. There are however different models for the HTC Vive controllers and Oculus Rift controllers. The controller model the user themselves sees is also displayed to others. Just like a user's own controller, there are also laser rays attached to other users' controllers.

The minimalist user visualization (user model) we implemented is enough to see where a user is positioned in the virtual room. The rotation of the HMD lets one see in which direction the user looks. The laser rays attached to the controllers let one see where

4.1. Frontend

and at which object a user points their controllers at. This enables the possibility to draw another user's attention to a system, e.g., to the visualized landscape. This may be helpful, since highlighting in the landscape is currently not an option.

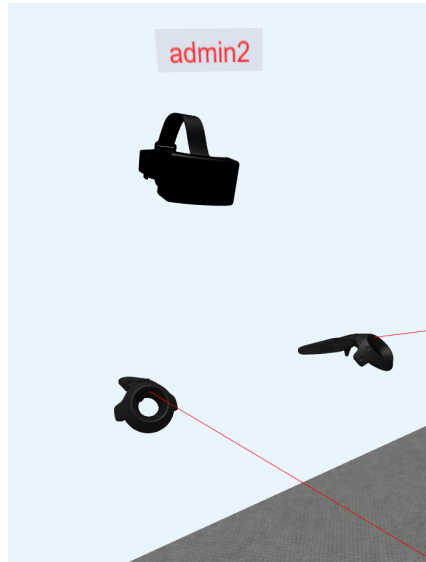


Figure 4.4. The user representation consisting of HMD, two controllers and name tag

Name Tag

Identifying the mapping between users and models in the virtual room may be difficult if multiple users are connected. While this can be communicated when in the same physical room or using VoIP, a visual indicator showing the user's name may come in handy.

Above the HMD, we add the name of the user in their respective color, as seen in Figure 4.4. Our first approach for this uses a `THREE.Sprite` plane. These objects always face toward the camera and match the camera's rotation. That way they are always readable when in sight. However, a user's name tag may be rotated in a way that it is partially inside the HMD and not readable anymore. Furthermore, we find that `THREE.Sprites` are quadratic and thus do not meet the aspect ratio for a name.

Therefore, we decide to use a `THREE.PlaneGeometry`, which is a flat shaped geometry with only two sides. Its size can be matched with the length of the user's name. To actually display the name on this geometry, we need a `THREE.Material`. This contains the image to display. Using a `HTML Canvas`, the name is added to it.

4. Implementation

4.1.6 Controls

In this section, we explain the changes and addition made to the assignment of keys on the Oculus Touch. A draft of the Oculus Touch controllers can be seen in Figure 4.5. The numbers assigned to the buttons in this figure, will be used for easier reference in the following.

Options Menu

The options menu is opened by once pressing ❶. One uses the laser ray on the right controller to interact with the menu. Pointing it at one of the submenu buttons will highlight the corresponding button. Pressing ❸ on the right controller whilst pointing at a submenu button, will open it. Navigating back through menus can either be done by clicking at the back button in the corresponding submenu or by pressing ❶. One closes the options menu by either clicking at the Exit button or by hitting ❶.

Move Application

In VR1.1 applications are moved by pointing at them and pressing one of the analog sticks on the Oculus Touch controllers. However, during development this assignment of keys felt rather unnatural and the button is a little harder to press. We, therefore, decided to put this functionality on ❷ (right). It is also worth noting that the grip buttons are the recommended buttons for picking up and releasing objects, according to Oculus VR.³

Text Box

The text box is the window that can be opened to display more details on a selected object, e.g., component or class. It looks like a notepad. In VR1.1 this window is opened by clicking the left of right grip button (❹). If one presses it on the left controller, it will be added to the right side of the left controller. If one presses it on the right controller, then it is displayed on the left side of the right controller. Since we decided to put the moving of applications on ❷ (right), we need to put the text box on another button. For simplicity, we put this action on ❺. Furthermore, we removed the text box from the left controller, since it can overlap with the options menu.

User List

The user list is a new feature and thus still needs a button to open. The button ❷ (left) is not assigned, since we removed the text box functionality from it. During development, we decided to make button assignments on the HTC Vive and Oculus Touch controllers as similar as possible. The HTC Vive, however, has fewer buttons. The only buttons still

³<https://developer.oculus.com/blog/tech-note-touch-button-mapping-best-practices/>

unassigned on the HTC Vive controllers are the Trackpad on the left controller, which is analogue to the analog stick on the Oculus Touch, and the grip button on the left controller (②). Since ② (left) is easier to press, we put the user list on this button. Pressing and holding it will open and keep open the user list until the button is released. Releasing it will close the user list.

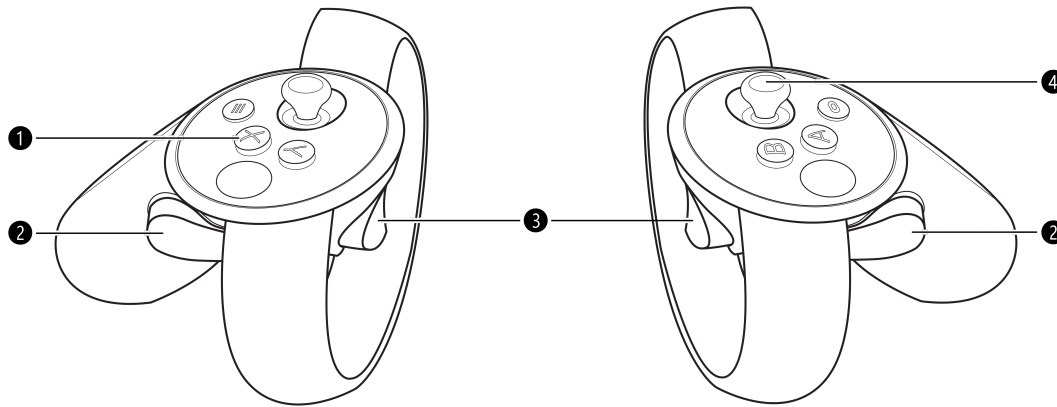


Figure 4.5. Draft of the Oculus Rift controllers with numbers for key referencing

4.2 Backend

In this section the implementation in the VR backend extension is explained. The implementation is found in the corresponding GitHub repository⁴. Our implementation is based on a dummy extension.

4.2.1 Main Structure

The main class used for implementation we named `MultiUserMode`. For the WebSocket connection we use a Java WebSockets⁵ library. It contains a class called `WebSocketServer`, which our `MultiUserMode` class inherits from.

The main entry point for an extension is the `SetupListener` class and, more precise, the method `contextInitialized`. Inside it, we create a new instance of our `MultiUserMode` and call its `start` method. Inside the constructor, a new WebSocket server is created. The `start` method creates a new Thread running the main loop.

⁴<https://github.com/ExplorViz/explorviz-backend-extension-vr/tree/v1.0>

⁵<https://github.com/TooTallNate/Java-WebSocket>

4. Implementation

4.2.2 Main Loop & Messages

The backend extension has one main loop. Its purpose is to make sure the server's tick rate runs at 90 frames per second. This corresponds to the refresh rate of both the HTC Vive and Oculus Rift. The main loop's only job is to call the tick method 90 times a second, which sends out data/messages to all the clients.

Messages for the clients are not immediately sent out, but stored in separate queues first. Those queues are realized as `JSONArray`'s, which allow easy appending of elements. Using `JSONArray`'s right away saves the conversion to other data structures every time data is sent, since `JSONArray` is already JSON compatible.

Data from a client is received in the `onMessage` method. We use the benefits of concurrency and create a new thread for every incoming message. This thread then decides what to do with the received data based on the value corresponding to the event key inside the message.

4.2.3 Synchronization

The backend extension stores the current state of the landscape. When the first user connects, the backend extension requests the latest landscape from the backend and stores it. Furthermore, a `HashMap` is instantiated, mapping system ids, respectively nodegroup ids, to a boolean value each. This value stores whether a system or nodegroup is open or closed. A new client is sent a message containing exactly the info inside that `HashMap`. That way, the landscape of the client can be set to the server's state. Furthermore, the backend extension saves the position and rotation of the landscape, as it can be moved by the users. Applications are stored in a similar fashion. When a user connects, a new user object is created which stores information such as the id, name, connection state, color or what controllers the user uses. To keep track of all the user objects, they are stored in a `HashMap`, linking the user's id to the corresponding user object.

Evaluation

5.1 Goals

To the best of knowledge, there currently do not exist any tools for software visualization that feature a collaborative VR mode. Therefore, we want to carry out a first qualitative test of usability. The aim of the study is not a comparison, e.g. with working together in front of a computer.

- ▷ G1: Intuitiveness and usability of the controls
- ▷ G2: Usability of the VR multi-user mode in general
- ▷ G3: Usefulness of visual illustrations
- ▷ G4: Usefulness of the spectator mode

5.2 Methodology

This section describes the methodology used for our experiment with test subjects. We want to check how well our extension is received by users in terms of usability. For this purpose, the test persons have to solve tasks using ExplorViz VR. They then complete a pre-printed questionnaire in which they evaluate their experience.

5.3 Experiment

In this section the conducted experiment is described. The survey including questionnaire is presented and both setup and procedure explained.

5.3.1 Survey

Personal Information

The first part of our survey is used to collect personal information about the test subjects. That includes data such as the *subject of study*. Furthermore, it includes 9 more questions

5. Evaluation

and statements as seen in Table 5.1. The test subjects rated them with 0, +, ++, and +++. 0 stands for no experience, do not have the condition or do not know the other test subjects at all.

Whereas +++ stands for very experienced, very strong condition or know the other test subject very well.

Table 5.1. Questions and statements of the general questionnaire

ID	Description
A1	Do you wear glasses?
A2.1	Do you have any visual impairment?
A2.2	If so, which:
A3	Experience with objectoriented programming
A4	Experience with ExplorViz
A5	Experience with VR
A6	Are you claustrophobic?
A7	Are you afraid of heights?
A8	Do you suffer from seasickness?
A9	How well do you know the other proband?

We are interested in the test subject's experience with

- ▷ object-oriented programming
- ▷ ExplorViz
- ▷ VR

since a test subject with higher experience might have an advantage over a test subject with little experience. Furthermore, these are skills expected by a real user using the ExplorViz VR extension.

Since VR is tolerated by people differently, we are interested the following conditions

- ▷ claustrophobia
- ▷ fear of heights
- ▷ seasickness

5.3. Experiment

During the experiment, two tests subjects use the VR mode collaboratively. Since our extension is expected to be used by software developers that know each other, we asked the test subjects how well they know each other.

Debriefing Statements

After solving the tasks, the test subjects are asked to rate their experience. The subjects are, therefor, given a list of statements that they should rate with either

- ▷ - - (strongly disagree)
- ▷ - (rather disagree)
- ▷ + (rather agree)
- ▷ ++ (strongly agree)

Below every statement is an empty line, which the test subjects can use to directly respond to the corresponding statement.

Debriefing Statements on Interaction

In Table 5.2, the statements regarding interaction including their id are displayed. We want to find out if the current navigation and movement is intuitive.

Table 5.2. Statements on interaction

ID	Description
B1.1	Moving and rotating the landscape is well realized
B1.2	The moving and rotating of the applications is well realized
B1.3	The movement (incl. teleportation) in the virtual space is intuitive
B1.4	Navigation through the menu is intuitive
B1.5	Highlighting objects is a useful feature

Debriefing General Statements

The statements seen in Table 5.3 regard the general usability and impression of the multi-user mode.

5. Evaluation

Table 5.3. General statements

ID	Description
B2.1	I had the impression that I was in the same room with the other user (positions and state of users, landscape & applications were synchronized)
B2.2	ExplorViz with VR extension is suitable for team work
B2.3	During the experiment (except spectator mode) I felt nausea or something alike
B2.4	I would use ExplorViz with the VR extension again

Debriefing Statements on Visual Appearance

We were interested in the subjects' opinions on the implemented text overlays, including menus, and user visualization. Statements regarding this, including id, are seen in Table 5.4.

Table 5.4. Statements on visual appearance

ID	Description
B3.1	The structure of the menu is intuitive
B3.2	The visualization of other users is well done
B3.3	The movements of the other user were displayed to me without delay
B3.4	The number of text insertions was reasonable
B3.5	Text overlays were clearly readable
B3.6	Text overlays were visually appealing (duration, length, position, color, size, animation)

Debriefing Statements on Spectating

Spectating is a new feature. Since the user loses control over their movements in VR, we want to investigate the perception and usefulness of this feature. The statements including their ids are shown in Table 5.5.

Table 5.5. Statements on spectating

ID	Description
B4.1	While spectating I felt nausea or something of the like
B4.2	I would use the spectator mode again

Further Comments

In the last part of the survey, the test subjects get the chance to write down any further comments they have regarding features, improvements, or more. The exact names of the free-text fields and their ids are displayed in Table 5.6.

Table 5.6. Further comments text fields of the survey

ID	Description
C1	Proposals for future features
C2	Improvement proposals
C3	Further remarks

5.3.2 Experimental Setup

This section describes the experimental setup used for the evaluation.

Hardware Configuration

For our experiment, we employ the three machines shown in Table 5.7. Machine 1 and 2 have identical hardware. Machine 1 runs the Oculus Rift and Machine 2 the HTC Vive. In our setup, the Oculus Rift is used with two Oculus Touch controllers (one left and one right) and three Oculus Sensors. The play area is configured to an almost quadratic area of approximately 4.2 m^2 . The HTC Vive uses two HTC Vive controllers and two base stations. The play area is configured to an area of approximately 14.4 m^2 .

5. Evaluation

Table 5.7. Computers used for evaluation

	Machine 1 & 2	Machine 3
CPU	Intel Core i5-6500	Intel Core i5-6500
GPU	NVIDIA GeForce GTX 1070	NVIDIA GeForce GTX 950
RAM	16 GB	16 GB

Software Configuration

All three machines run Microsoft Windows 10 Pro - 64 Bit. Machine 1 and 2 both run an instance of the ExplorViz frontend and have the ExplorViz frontend VR extension installed. Machine 3 has installed and runs an instance of the ExplorViz backend and ExplorViz backend extension VR. Both the frontend and the frontend VR extension are configured such that they connect to the backend and backend extension VR on Machine 3, respectively. During the training phase, an instance of the `kickerSampleApplication`¹ is also running on Machine 3.

5.3.3 Experiment Procedure

Evaluating the multi-user mode takes more than one test subjects. Therefore, two test subjects came together in each session. One subject used the `Oculus Rift` and the other one the `HTC Vive`. The choice of devices was left to the subjects. After the devices were assigned, the subjects were welcomed, briefly explained what the study was about and that all data was treated anonymously. The procedure was then briefly explained.

The test subjects then filled-in the general questionnaire regarding personal information. Furthermore, the subjects were informed that they could ask questions regarding the questionnaire at any time. Afterward, the subjects were orally introduced to ExplorViz.

Subsequently, the subjects were introduced to the controls. For this purpose, one instructor each explained the controls to the subjects in parallel. On the side of the `Oculus Rift`, the subjects were first explained the controls outside virtual reality. The test subjects took both controllers into their hands and were explained the meaning of the keys and which actions can be performed with them. Then the training phase was begun. On the `HTC Vive` side, this first explanation of the controls was not always carried out before the test phase, but the test phase was started directly.

Training Phase

At the beginning of the training phase, the test subjects put on the HMDs and pick up the controllers. On the basis of a small test landscape (`kickerSampleApplication`) the test

¹<https://github.com/czirkelbach/kickerSampleApplication>

persons learn the controls within VR. The controls are reviewed with them once again and they are encouraged to perform all possible interactions once. They are explained the different views (landscape & application) that exist within ExplorViz. The test phase lasts until the test subjects are reasonably confident in the controls. To make sure all controls were explained, the instructors had a list containing all controls. It is found in *Appendix B*.

The test subjects keep the HMD on and the instructors change the landscape in the backend to the ExplorViz dummy landscape. The browser tab, running the ExplorViz frontend, is refreshed on both the computers the HMDs are connected to. The dummy landscape is then ready for use in VR. The main phase begins.

Main Phase

In the main phase, the test subjects put into practice what they have learned. They are read tasks that they are to solve alone or together. Both test subjects are explicitly encouraged to talk with each other. First, the test persons both connect to the server. They are then encouraged to test certain functionalities such as adjusting their own height or opening the user list. The test subjects search together for nodes and applications in the landscape and are to find certain components in applications. The spectator mode is also used by both test subjects. A complete list of all tasks is found in *Appendix A*.

Rating Phase

After the main phase, both test subjects take off the HMDs and put down the controllers. Now they have to evaluate the software on their own and fill out the rating form. The test subjects are informed that 'text insertions' in statement B3.4 has the same meaning as 'text overlays' in B3.5 and B3.6. Furthermore it is explained that all overlays (hints, notifications) but also all menus are meant. It is also added that this does not refer to texts that appear in landscapes or applications. Furthermore, it is explained how the rating is to be understood. This means that - - does mean you do not agree with the statement and ++ means that you strongly agree with the statement. Furthermore, it is explained that the blank line under each statement can be used to leave a direct comment on it. The test subjects can ask further questions regarding comprehension.

5.4 Results

In this section the results of the study are presented. For that, we assign integer values to the evaluation units of our questionnaires as seen in Table 5.8. This allows to work with the data better and to calculate e.g. mean value and standard deviation.

5. Evaluation

Table 5.8. Mapping evaluation units to inter values

Evaluation Unit	Integer Value
--	-2
-	-1
0	0
+	1
++	2
+++	3

5.4.1 Test Subjects

Altogether 22 test subjects participated in our study, which equals 11 groups of 2 persons each. 14 subjects study computer science or business informatics in the bachelor's programme. 4 persons stated to be researchers. The other 4 stated that they were either studying a different subject or pursuing a different profession. The average user matches the following profile according to Table 5.9

- ▷ well experienced in object-oriented programming (A3)
- ▷ no experience with ExplorViz (A4)
- ▷ no or little experience with VR (A5)
- ▷ not claustrophobic (A6)
- ▷ is a little afraid of heights (A7)
- ▷ does not suffer from seasickness (A8)
- ▷ knows the other participant quite well (A9)

5.4. Results

Table 5.9. Results of the general questions

ID	Mean	SD
A3	2.0	0.9258
A4	0.3636	0.9021
A5	0.7273	1.0320
A6	0.0455	0.2132
A7	0.9091	1.0193
A8	0.4545	0.7385
A9	1.9091	0.8679

5.4.2 Interaction Results

In Table 5.10 the results of the analysis of data on interaction are displayed.

Table 5.10. Results of the interaction statements

ID	Mean	SD
B1.1	1.136	1.246
B1.2	1.773	0.429
B1.3	1.727	0.703
B1.4	1.455	0.596
B1.5	1.591	0.908

5.4.3 General Results

Table 5.11 displays the results of the general rating part. Its focus is the overall opinion on the multi-user mode and the physical comfort during use.

5. Evaluation

Table 5.11. Results of the general statements

ID	Mean	SD
B2.1	1.909	0.294
B2.2	1.545	0.739
B2.3	-1.773	0.685
B2.4	1.364	1.093

5.4.4 Visual Appearance Results

The visual appearance is mainly characterized by the user representation and visual overlays. In Table 5.11 the results of the statements regarding visual appearance are shown.

Table 5.12. Results of the visual appearance statements

ID	Mean	SD
B3.1	1.545	0.510
B3.2	1.136	0.990
B3.3	1.682	0.477
B3.4	1.727	0.703
B3.5	1.045	0.950
B3.6	1.045	1.133

5.4.5 Spectate Feature Results

The spectate feature is considered separately, because it is rather experimental. In Table 5.11 the results of the statements regarding it are displayed.

Table 5.13. Results of the spectating statements

ID	Mean	SD
B4.1	-0.773	1.412
B4.2	0.682	1.460

5.5 Discussion

In this section, we discuss the results of the experiment presented in Section 5.4.

5.5.1 Interaction Results

The Statements B1.2, B1.3 and B1.5 were rated particularly positively, with an average value of over 1.5 for all of them. B1.1 was rated worst with an average of 1,136. Here, however, the standard deviation is very high. This results from the fact that 4 of the 22 test persons did not agree with the statement at all or rather not. Two subjects remarked that the movement of the landscape should be realized just like moving applications. That way the user could grab onto the landscape and position it more easily. The current menu for moving the landscape requires the user to press buttons over and over. Moving thus demands some time. Furthermore, moving landscapes different than applications might be less intuitive.

5.5.2 General Results

B2.1 achieved the best results overall. With a score of 1.9 it suggests that the impression of being in the same room with the other user was great. The tolerability of the VR mode was confirmed by B2.3. Outside the spectator mode the users tolerated the mode well. Most users also approved that the developed multi-user mode is well suited for working in a team. Rather controversial, even though it achieved a good rating, is B2.4. With a SD of 1.1, there were 3 test subjects who would rather not use the VR mode again. The question arises how the statement was understood, since the statement was completely agreed by all subjects who are not active in the field of computer science. We therefore suspect that some subjects felt asked about their overall liking of ExplorViz VR while others might have answered the statement more depending on whether they would actually use it at work. The overall general impression is in conclusion good.

5.5.3 Visual Appearance Results

We here discuss the results of the statements regarding visual appearance, which are seen in Table 5.12. With a rating of over 1.5, the intuitiveness of the options menu was confirmed. However, it was noted by one subject that navigating through the menu using the analog sticks should be an option. The current implementation requires the user to use the right controller and point at the buttons in the menu.

B3.3 was rated by all test subjects with either 'rather agree' or 'strongly agree'. The test subjects thus all had the impression that the other person's actions in VR were mostly delay free.

The text overlays (B3.5, B3.6) were somewhat controversial. Although the test subjects were told that the labels in the landscape and applications were not meant, there were comments about labels in the empty fields of B3.5 and B3.6. Therefore, we do not know

5. Evaluation

whether the statement was answered incorrectly. It was further noted that the text inserts of some test subjects were difficult to read. We cannot make sure that the test subjects had the HMD correctly mounted and may have experienced limitations in vision as a result.

5.5.4 Spectate Feature Results

In the following we refer to the results in Table 5.13.

The specate feature is the most controversial feature we introduced. B4.1 and B4.2 suggest that over 75 percent of test subjects would use the specate feature again. However, this means that about a quarter would not. Roughly 32 percent of people felt nausea or something of the like during the short use of the spectate feature.

One correlation we noticed is that all test subjects that felt nausea or alike during spectate are at least a little afraid of height. Even if we cannot derive any causality from the low number of subjects, we recommend observing this correlation in further studies.

5.6 Improvement Suggestions

We received plenty of feedback by the test subjects in the form of comments. These are found in *Appendix C*. The top five stated improvement suggestions are shown in Figure 5.1.

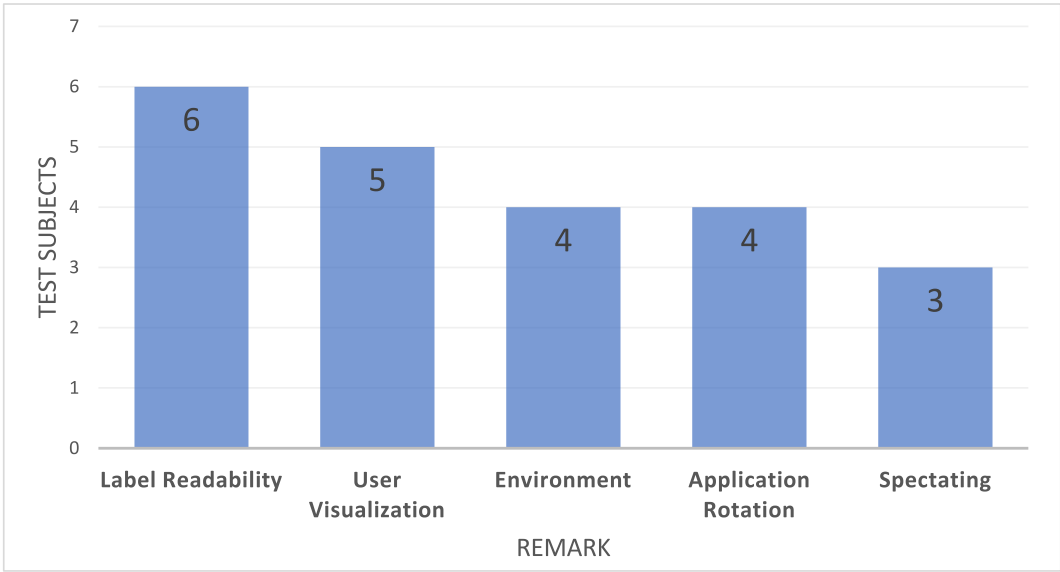


Figure 5.1. Top 5 remarks by the test subjects

The most commented is the label readability. A little over 25 percent of the subjects

5.7. Threats to Validity

found the labels in the landscape or applications too small, blurred or didn't like the fact that long labels are cut off. Latter happens in applications. That the font of labels is rather small has to do with the fact that, especially in the case of classes, the boxes are also small. Making labels larger or not cutting long label off might result in labels overlapping. One test subject therefore suggests enlarging the component's label and showing its entire name when the user specifically points at it.

Another approx. 25 percent of subjects suggest the enhancement of user visualization. The current user model consists of only a HMD model and two controller models. Overall, the subjects want a better representation that includes a body. Suggested were stick person bodies or avatars for colleagues.

A little under 20 percent would like to work in a different virtual environment. The current design is very plain. The background color is a light blue and the floor texture is also rather simple. To enhance the user experience, subjects suggested the improvement of colors and a more beautiful background. One even suggested a beach setting.

The same amount of people suggest the improvement of rotating apps. Currently, applications are bound to the user's controller and the application is moved by moving the controller accordingly. Test subjects propose to rotate the application by moving the analog stick.

A little less than 15 percent of test subjects doubt the usefulness of the spectate mode. Since users can always position themselves next to each other and a user can not do anything but watch while spectating, its added value is questioned.

5.7 Threats to Validity

In this section we discuss possible threads to validity of the experiment's results.

5.7.1 Test Subjects

22 people took part in the experiment, which corresponds to 11 groups of two. As the number of groups is small, the data should be treated with caution. To validate the data, a follow-up study with at least 30 groups of two should be performed.

Furthermore, multiple test subjects were personally known to one or both the instructors. Therefore, it can not be ruled out that they may have been influenced in their judgement. Moreover, some of the test subjects were neither students nor researcher in the field of computer science or business informatics.

5.7.2 Introduction to Controls

The test subjects using the Oculus Rift were introduced to the controls differently than the subjects using the HTC Vive. While former subjects got an introduction to the controls before even putting on the HMD, the latter subjects were mostly given the controls after putting

5. Evaluation

the HMD on. We think that the subjects using the Oculus Rift thus had an advantage, since they could physically see and test out the controllers including their buttons. Due to the limitation in resolution of both devices, the controllers inside ExplorViz VR are a little blurry and key labels are partly unreadable. We therefore suspect learning the controls inside ExplorViz to be more difficult. For further studies, we suggest a consistent introduction.

5.8 Conclusion

All test groups were able to solve the tasks, although some needed small hints regarding controls. The goal regarding intuitiveness and usability of the controls (G1) was reached, as all statements were on average rated at least with 'rather agree'. The usability of the multi-user mode (G2) is also reached, since the response was as well rather positive. Overwhelmingly rated was that the users really felt like they were together in the virtual room. Even though text overlays were only moderately well received, the visual appearance was rated very positively overall. Thus goal G3 is also reached. However, the user appearance has potential for improvement. Very mixed opinions are raised by the speculate feature. Since on average not even a 'rather good' was achieved, we consider G4 not achieved. The paper-based questionnaire and all raw results are found in *Appendix A* and *Appendix C*, respectively.

Related Work

[Schepper et al. 2015] present a VR multiplayer game that can be played with two users at a time. The users are represented by an avatar in the virtual room. That way they can see each other. Additionally, a third user can spectate the game from a bird's eye perspective. Goal of the game is to capture a flag that is hidden inside a maze.

The users wear a Oculus Rift Development Kit 2 (see Figure 6.1 left). It allows the user to see the virtual world and to move their head to look around. The tracking used for position detection works by setting up 4 smartphones and using their cameras. The setup of a smartphone is displayed in Figure 6.1 (right). They are covering a defined area. The frames captured by them get processed by a special algorithm, which calculates and determines the position of the user. This position gets then sent to a server, which distributes the data to other users. The architecture used is a star topology. There is one server and multiple clients. The protocols used for communication are TCP and UDP. UDP is used for position data. If one position does not arrive, but the next one does, latter will make former obsolete. Most sent data is position data. However, some data for starting and setting up the game and picking up the flag is necessary. Since this data has to arrive, the TCP protocol is used here. While the position of the user can be tracked, the user's rotation cannot. Therefore the user will hold a computer mouse in one hand, which allows them to rotate and shoot at the opponent.

Two main problems occurred during development. Firstly, the user can walk into, but not through walls, in the virtual room. There is a collision detection. However, in the real world the user can walk further and further, even outside the tracking area. Another issue is the portability. There was no good way of realizing portability but by equipping the user with a notebook in a backpack. Even the tracking realized by using smartphone cameras comes at a price. The average framerate is between 2.80 FPS on the Motorola Moto G and 10.46 FPS on the Samsung Galaxy S5.

Unlike in our project, a multiplayer game is implemented in [Schepper et al. 2015]. In contrast, ExplorViz is a software to use at work. Instead of tracking with smartphone cameras, the Oculus Rift Consumer Version 1 comes with two sensors. These have more accurate tracking, but must be connected to the computer via USB and can only cover a very limited area. Like Schepper et al. 2015, we use a star topology and have a central node that collects data from the clients and distributes it to the others. Position data is transferred via UDP in the VR multiplayer game presented by [Schepper et al. 2015]. This

6. Related Work



Figure 6.1. User equipment (left) and smartphone attached to a chair (right) [Schepper et al. 2015]

provides a lower overhead than TCP and lost packets are not retransmitted. This is not possible in a web application like *ExplorViz*, because current browsers do not support UDP for security reasons. However, UDP is an advantage.

Conclusions and Future Work

This chapter summarizes the thesis and the conducted experiment. Furthermore, an outlook is given regarding the future of the VR extension, also with regard to the developed multi-user mode.

7.1 Conclusions

In this bachelor's thesis, we developed the existing ExplorViz VR extension further. What used to be a pure frontend extension has been expanded by an additional extension in the backend. The newly developed multi-user mode provides users with the ability of meeting in virtual reality and exploring software landscapes in a collaborative way. A minimal user representation consisting of HMD and controllers allows to see others and especially their movements in the virtual world. All users can manipulate the software landscape together, open multiple applications and work with them. The new backend extension stores the current state, the necessary user data and keeps the frontend extensions synchronized. In order to exchange data between the two extensions, we decided on a WebSocket connection. Furthermore, new features like menus were added to improve the user experience through customizations. If one prefers to continue working alone, the ExplorViz VR frontend can still be used independently.

In our study, 11 groups of two tested the collaborative mode and subsequently evaluated it. The mode was generally well received and it was confirmed that it is suitable for working in a team. Users have the feeling of being together in the virtual world and perceiving the movements of others without much delay. Moving around in the virtual world by either physical movement or teleportation was considered intuitive. Likewise, navigation in the virtual world was accepted as intuitive and well realized.

7.2 Future Work

New ideas arose during the development of the collaborative mode. Test subjects also provided proposals for improvement as part of the evaluation. These are described and discussed here.

7. Conclusions and Future Work

When systems are opened or closed, the landscape is layouted from scratch. This causes the image on the HMD display to either freeze or turn black. In collaborative mode, this happens every time anyone changes the landscape and this can be irritating. JS uses only one thread for web applications, however, there are ways to circumvent this. Loading landscapes in another thread will likely increase the user experience.

Many test subjects requested that the user receives a body. We stand by our minimal approach, but can understand that this feature is desired. The user could feel even more comfortable.

An extended control of the applications was also desired. The rotation of the applications using the analog stick was explicitly mentioned here. This same idea came to us during development, but our attempts failed. We could also imagine moving the application towards and away from oneself using the analog stick instead of rotating it.

The control of the VR mode especially in menus is designed rather for right-handers. In the future the possibility could be introduced to swap the functions of the controllers in order to make it easier for left-handers to use the VR mode.

At the moment there is no way to look up the control in VR mode yourself. We could imagine a tutorial that would guide the user through the controls using a sample landscape in order to learn it. Also possible would be a simple menu in which the keys are explained. So the user could look up the keys if they do not yet know them or have forgotten something.

11 groups participated in the conducted study. We consider this to be too little to make a valid statement about the usability of the collaborative mode. In the future, another study should be conducted in which at least 30 two-person groups participate.

Bibliography

- [Anthes et al. 2016] C. Anthes, R. J. García-Hernández, M. Wiedemann, and K. D. State of the Art of Virtual Reality Technology. In: *2016 IEEE Aerospace Conference*. Mar. 2016. (Cited on page 3)
- [Farahani et al. 2016] N. Farahani, R. Post, J. Duboy, I. Ahmed, B. J. Kolowitz, T. Krinchai, S. E. Monaco, J. L. Fine, D. J. Hartman, and L. Pantanowitz. Exploring virtual reality technology and the Oculus Rift for the examination of digital pathology slides. *Journal of Pathology Informatics* 7.1 (2016). (Cited on page 3)
- [Fittkau et al. 2015a] F. Fittkau, E. Koppenhagen, and W. Hasselbring. Research Perspective on Supporting Software Engineering via Physical 3D Models. In: *IEEE 3rd Working Conference on Software Visualization (VISOFT 2015)*. IEEE, Sept. 2015. (Cited on page 6)
- [Fittkau et al. 2015b] F. Fittkau, A. Krause, and W. Hasselbring. Exploring Software Cities in Virtual Reality. In: *IEEE 3rd Working Conference on Software Visualization (VISOFT 2015)*. IEEE, Sept. 2015. (Cited on page 4)
- [Fittkau et al. 2017] F. Fittkau, A. Krause, and W. Hasselbring. Software Landscape and Application Visualization for System Comprehension with ExplorViz. *Information and Software Technology* 87 (2017). (Cited on page 4)
- [Fittkau et al. 2013] F. Fittkau, J. Waller, C. Wulf, and W. Hasselbring. Live Trace Visualization for Comprehending Large Software Landscapes: The ExplorViz Approach. In: *1st IEEE International Working Conference on Software Visualization (VISOFT 2013)*. Sept. 2013. (Cited on page 4)
- [Hansen 2018] M. Hansen. Collaborative Software Exploration with the HTC Vive in ExplorViz. Bachelor thesis. Kiel University, Sept. 2018. (Cited on page 2)
- [Häsemeyer 2017] T. Häsemeyer. Kollaboratives Erkunden von Software mithilfe virtueller Realität in ExplorViz. Bachelor thesis. Kiel University, Sept. 2017. (Cited on pages 1, 8, 11)
- [Krause 2015] A. Krause. Erkundung von Softwarestädten mithilfe der virtuellen Realität. Bachelor thesis. Kiel University, Sept. 2015. (Cited on pages 1, 4)
- [Qigang and Sun 2012] L. Qigang and X. Sun. Research of Web Real-Time Communication Based on Web Socket. 05 (Jan. 2012). (Cited on page 12)
- [Schepper et al. 2015] T. D. Schepper, B. Braem, and S. Latre. A Virtual Reality-based Multiplayer Game using Fine-grained Localization. In: *2015 Global Information Infrastructure and Networking Symposium (GIIS)*. Oct. 2015. (Cited on pages 47, 48)

Bibliography

- [Sutherland 1968] I. E. Sutherland. A Head-mounted Three Dimensional Display. In: *Proceedings of the December 9-11, 1968, Fall Joint Computer Conference, Part I*. ACM, Dec. 1968. (Cited on page 3)
- [Zirkelbach et al. 2018] C. Zirkelbach, A. Krause, and W. Hasselbring. On the Modernization of ExplorViz towards a Microservice Architecture. In: *Combined Proceedings of the Workshops of the German Software Engineering Conference 2018*. CEUR Workshop Proceedings, Feb. 2018. (Cited on page 12)

Appendix A

Dear participant,

we thank you very much for your participation in this experiment. You and another participant are about to test and evaluate a collaborative virtual reality (VR) extension to ExplorViz. You are free to quit the evaluation whenever you are not feeling comfortable anymore. First of all, we would like you to answer some general questions about you on the next page. This will allow us to add some context to the results later on. Your answers and test results are anonymous. Then we are proceeding by introducing you to ExplorViz and the input devices for VR. When you have learned the basics about ExplorViz and the controls you will be asked to solve some exercises together with your partner. Lastly, you will be asked to rate certain aspects about the user experience. This will help us improve the extension and tell us how well ExplorViz can be used with multiple users in virtual reality.

1 General Personal Data

ID: *Vive* ☐ *Rift* ☐

Age:

Profession: *student* ☐ *researcher* ☐ *other* ☐

Subject of study: *bachelor* ☐ *master* ☐

Gender: *male* ☐ *female* ☐ *diverse* ☐

Do you wear glasses? *yes* ☐ *no* ☐

Do you have any visual impairment? *yes* ☐ *no* ☐

If so, which:

	0	+	++	++ +
Experience with objectoriented programming	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Experience with ExplorViz	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Experience with VR	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Are you claustrophobic?	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Are you afraid of heights?	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Do you suffer from seasickness?	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
How well do you know the other proband?	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

2 Introduction

ExplorViz is a monitoring and visualization software for large software landscapes. ExplorViz uses two different views for the visualization which are shown simultaneously with the VR extension. The landscape view is a view of a software landscape and is particularly suitable to get an overview of landscapes. Here you can see systems (grey), servers (green) and the software running on the servers (blue). The communication between software is represented by orange lines, where the thickness of the lines correlates with the the number of calls it represents.

The application view represents a three-dimensional model of the software. On top of a grey foundation software packages (components) are shown in green, which in turn can contain components or individual classes (blue). The height of the blue blocks indicates the number of objects belonging to the class. Here, too, the communication between objects is visualized with orange lines. You can select individual classes or call up additional information for a class.

This type of representation is intended to be a metaphor for a three-dimensional city, with the classes here representing (high-)buildings and communication between classes are streets.

Now that you are familiar with the concepts we would like you to get familiar with the controls. On the next page there is an overview of the functionalities of all buttons on the controllers you are about to use. Please use the HTC Vive or Oculus Rift respectively now while we guide you through all control options. Feel free to ask questions throughout the experiment of something is unclear to you.

3 Tasks

1. Connect via the menu.
2. Change your height as you like.
3. Open the user list.
4. Move and rotate the landscape as you like.
5. Find the node '10.0.2.2'.
6. Find the application 'Wiki' and try to open it.
7. With how many applications does 'Webshop' communicate?
8. Open the application 'Webshop'.
9. Move and rotate the application 'Webshop' as you like.
10. Mark the class 'ItemHelper' which is part of the application 'Webshop'.
11. How many active instances has the class 'ImplementationHandler'
(located in component org/webshop/kernel/impl)?
12. Open a second application.
13. Use the spectate feature (only as long as you are comfortable using it).
14. The spectated proaband may mark the component labeling in Webshop.
15. Quit spectating and spectate each other with reversed roles.
16. Mark component connector in DatabaseConnector.
17. Quit spectating.
18. Close all open system and components.
19. Disconnect via the menu.

4 Rating

	--	-	+	++
Moving and rotating the landscape is well realized	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
.....				
The moving and rotating of the applications is well realized	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
.....				
The movement (incl. teleportation) in the virtual space is intuitive	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
.....				
The structure of the menu is intuitive	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
.....				
Navigation through the menu is intuitive	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
.....				
The visualization of other users is well done	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
.....				
The movements of the other user were displayed to me without delay	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
.....				
I had the impression that I was in the same room with the other user (positions and state of users, landscape & applications were synchronized)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
.....				
The number of text insertions was reasonable	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
.....				

	--	-	+	++
Text overlays were clearly readable	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
.....				
Text overlays were visually appealing (duration, length, position, color, size, animation)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
.....				
Highlighting objects is a useful feature	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
.....				
ExplorViz with VR extension is suitable for team work	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
.....				
During the experiment (except spectator mode) I felt nausea or something alike	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
.....				
While spectating I felt nausea or something of the like	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
.....				
I would use the spectator mode again	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
.....				
I would use ExplorViz with the VR extension again	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
.....				

Proposals for future features

--

Improvement proposals

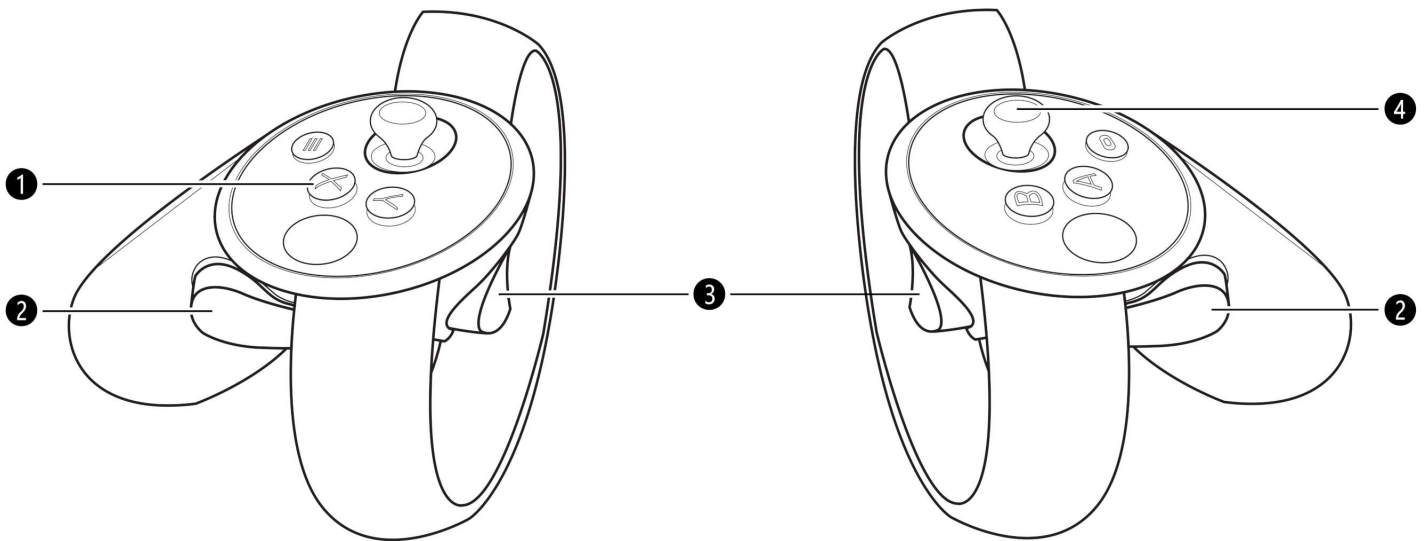
--

Further remarks

--

Appendix B

Rift Controllers:



You can target many objects in the virtual environment with the ray of the controller and interact with them through corresponding buttons. The ray of the left controller is colored black and that of the right one is colored green.

❶: (Left Controller):

Press this button to open the options menu. If in a menu, pressing the button can be used to navigate back through previous menus.

❷: (Left Controller):

Hold this button down to display a list of users connected to the server. Release the button to close the list.

❷: (Right Controller):

Target a 3D application with the ray of the controller and keep this button pressed to bind the 3D application to the controller. The application now follows all movements of the controller. Release the button to stop this behavior.

❸: (Left Controller):

Target the ground with the ray of the left controller and press this button to teleport yourself to the displayed circle on the ground. Target the red "X" above a 3D application with the ray of the controller and press this button to delete the 3D application. This button can also be used to select targeted clazzes and closed packages of a 3D application. Consequently the selected entity is colored red and the associated communication lines are highlighted. If nothing is targeted press this button again to unselect the entity and restore its color and the communication lines.









❸: (Right Controller):

Press this button to open/close targeted systems, nodegroups, packages and create 3D applications out of targeted 2D applications. Target the red "X" above a 3D application with the ray of the controller and press this button to delete the 3D application. This button can also be used to navigate through menus.

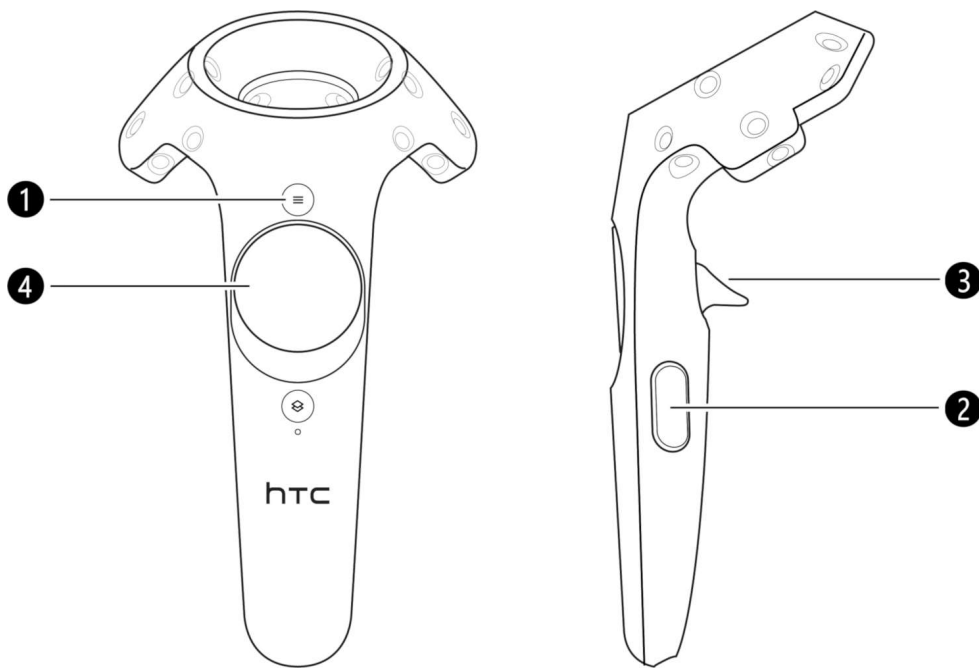
④: (Right Controller):

Press this button to display information about the targeted entity.

Keyboard:

- : Move the camera upwards
- : Move the camera downwards
- : Move the camera leftwards
- : Move the camera rightwards
- : Move camera forwards (Zoom in)
- : Move camera backward (Zoom out)
- : Rotate the environment forwards
- : Rotate the environment backwards

Vive Controllers:



You can target many objects in the virtual environment with the ray of the controller and interact with them through corresponding buttons. The ray of the left controller is colored black and that of the right one is colored green.

❶: (Left Controller):

Press this button to open the options menu. If in a menu, pressing the button can be used to navigate back through previous menus.

❷: (Left Controller):

Hold this button down to display a list of users connected to the server. Release the button to close the list.

❸: (Right Controller):

Target a 3D application with the ray of the controller and keep this button pressed to bind the 3D application to the controller. The application now follows all movements of the controller. Release the button to stop this behavior.

❹: (Left Controller):

Target the ground with the ray of the left controller and press this button to teleport yourself to the displayed circle on the ground. Target the red "X" above a 3D application with the ray of the controller and press this button to delete the 3D application. This button can also be used to select targeted clazzes and closed packages of a 3D application. Consequently the selected entity is colored red and the associated communication lines are highlighted. If nothing is targeted press this button again to unselect the entity and restore its color and the commuincation lines.









③: (Right Controller):

Press this button to open/close targeted systems, nodegroups, packages and create 3D applications out of targeted 2D applications. Target the red "X" above a 3D application with the ray of the controller and press this button to delete the 3D application. This button can also be used to navigate through menus.

④: (Right Controller):

Press this button to display information about the targeted entity.

Keyboard:

- : Move the camera upwards
- : Move the camera downwards
- : Move the camera leftwards
- : Move the camera rightwards
- : Move camera forwards (Zoom in)
- : Move camera backward (Zoom out)
- : Rotate the environment forwards
- : Rotate the environment backwards

Appendix C

ID	Vive/Rift	Age	Profession	Subject	Gender	Glasses?	Visual Impairment	Which?	OO	ExplorViz	VR	Claustro	Heights	Seasickness	Well know
1	Vive	24	student	student teacher	female	no	no		0	0	0	0	0	0	3
1	Rift	23	student	Biology	female	yes	yes	short	0	0	1	0	0	1	3
2	Vive	32	researcher	CS	male	yes	yes	short	2	3	2	0	2	2	2
2	Rift	28	researcher	CS	male	yes	yes	red/green	2	3	2	0	2	2	3
3	Vive	54	researcher	CS	male	no	no		2	1	1	0	0	0	2
3	Rift	39	researcher	CS	male	yes	yes	short	3	1	0	0	1	0	2
4	Vive	20	student	CS BA	male	no	yes	slight red/green	2	0	0	0	0	0	1
4	Rift	24	student	CS BA	male	yes	yes	short	2	0	0	0	1	0	1
5	Vive	22	student	CS BA	male	no	no		3	0	0	0	0	0	1
5	Rift		student	CS BA	male	no	no		2	0	1	0	2	0	1
6	Vive	22	student	CS BA	male	yes	yes	short	3	0	0	0	1	0	1
6	Rift	22	student	business CS BA	male	no	no		2	0	0	0	0	1	1
7	Vive	28	student	business CS BA	male	no	no		3	0	0	0	0	0	1
7	Rift	21	student	CS BA	male	no	no		3	0	3	0	1	1	1
8	Vive	22	student	CS BA	male	yes	yes	short	2	0	2	0	3	0	3
8	Rift	19	student	CS BA	female	yes	yes	long	2	0	0	0	0	0	3
9	Vive	22	student	CS BA	male	no	no		2	0	0	1	1	0	2
9	Rift	22	student	CS BA	male	no	no		2	0	0	0	0	0	1
10	Vive	22	student	CS BA	male	no	no		3	0	1	0	0	1	3
10	Rift	22	student	CS MA	male	no	no		2	0	3	0	1	0	3
11	Vive	30	other		male	yes	yes	short, 90% stereoscopic vision	2	0	0	0	2	2	2
11	Rift	26	student	history / philosophy BA	male	no	no		0	0	0	0	3	0	2
AVERAGE		25.9048							2.0000	0.3636	0.7273	0.0455	0.9091	0.4545	1.9091
SD		7.9618							0.9258	0.9021	1.0320	0.2132	1.0193	0.7385	0.8679

	#On Paper:	1	2	3	5	12	8	13	14	17	4	6	7	9	10	11	15	16
ID	Rift/Vive	B1.1	B1.2	B1.3	B1.4	B1.5	B2.1	B2.2	B2.3	B2.4	B3.1	B3.2	B3.3	B3.4	B3.5	B3.6	B4.1	B4.2
	1 Vive	2	2	2	1	2	2	2	-2	2	1	2	2	2	1	0	-2	2
	1 Rift	2	2	2	1	1	2	1	-2	2	1	1	1	2	1	1	-1	2
	2 Vive	1	1	2	2	2	2	2	-2	2	2	1	1	2	1	1	1	1
	2 Rift	2	2	2	2	2	2	-1	-2	-1	2	2	2	2	-1	2	1	1
	3 Vive	2	2	2	2	1	2	2	-2	2	1	1	2	2	2	2	-2	2
	3 Rift	2	1	2	1	2	2	2	-1	2	1	-1	2	2	2	-1	-2	
	4 Vive	1	2	1	1	2	2	2	-2	1	1	2	1	2	1	2	-2	1
	4 Rift	2	2	2	1	-2	2	1	-2	-1	1	-1	1	1	1	1	-2	-2
	5 Vive	2	1	1	2	1	2	1	-2	1	1	2	2	1	2	1	-1	-2
	5 Rift	1	2	2	2	1	2	2	-2	2	2	1	2	2	1	2	1	1
	6 Vive	2	2	1	1	1	2	1	-2	2	2	1	2	2	1	2	1	1
	6 Rift	1	2	2	1	2	2	2	-1	2	1	1	1	2	1	1	-1	1
	7 Vive	1	1	2	1	2	2	1	-2	1	1	1	2	1	1	-1	-1	1
	7 Rift	-2	2	2	2	2	2	2	1	2	2	1	2	2	2	-1	1	-2
	8 Vive	-1	2	2	2	2	2	2	-2	2	2	-1	1	-1	-1	-1	1	-2
	8 Rift	2	2	2	2	2	1	2	-2	2	2	2	1	2	-1	2	-2	1
	9 Vive	2	2	2	2	2	2	2	-2	2	2	1	2	2	2	2	-2	2
	9 Rift	2	2	2	0	2	2	1	-2	0	1	1	2	2	2	2	-1	2
	10 Vive	1	2	2	2	2	2	2	-2	2	2	2	2	2	1	1	-2	2
	10 Rift	2	1	-1	1	2	2	1	-2	-1	2	2	2	2	1	2	-2	2
	11 Vive	-1	2	2	2	2	2	2	-2	2	2	2	2	2	2	2	-2	1
	11 Rift	-1	2	2	1	2	1	2	-2	2	2	2	2	2	1	1	2	-1
Rift		1.1818	1.8182	1.7273	1.2727	1.4545	1.8182	1.3636	-1.5455	1.0000	1.5455	1.0000	1.6364	1.9091	0.9091	1.0909	-0.5455	0.5455
Vive		1.0909	1.7273	1.7273	1.6364	1.7273	2.0000	1.7273	-2.0000	1.7273	1.5455	1.2727	1.7273	1.5455	1.1818	1.0000	-1.0000	0.8182
AVERAGE		1.1364	1.7727	1.7273	1.4545	1.5909	1.9091	1.5455	-1.7727	1.3636	1.5455	1.1364	1.6818	1.7273	1.0455	1.0455	-0.7727	0.6818
SDEV		1.2458	0.4289	0.7025	0.5958	0.9081	0.2942	0.7385	0.6853	1.0931	0.5096	0.9902	0.4767	0.7025	0.9501	1.1329	1.4119	1.4601

ID	Rift/Vive	Remarks
1	Vive	C2: ansprechenderer Hintergrund
1	Rift	C1: bessere Darstellung *stick person drawing* <- ungefähr so oder eine Figur nach Wahl z.B. Firmenmaskottchen C2: bessere Farben; schönerer Hintergrund
2	Vive	B3.6: overlay anchor different sometimes (menu vs. connected) C1: menu option: showing controller mapping (half-transparent) (help dialogue) C2: + readability of labels (components, classes, systems) + movement reset capability (via menu) + rotation of applications could improved C3: The VR-extension provides a "real" working together in the same room feeling, well done.
2	Rift	B1.1: würde gerne Controll-Stick ebenfalls für Rotation nutzen B3.2: show text field of other users? B3.5: Manche waren sehr am Rand und da konnte ich sie schwer lesen, z.B. 'no data' bei der Applikation C1: Rotation der Applikation mit dem Controllstick C2: Labels von Applikationen etc. besser lesbar machen C3: Sehr flüssig und insgesamt intuitiv. Aber vermutlich nicht für den echten Einsatz geeignet.
3	Vive	B1.3: nach einer gewissen Zeit B1.5: useful for what? B3.2: Stirchmännchen wäre gut C1: Avatar für Kollegen C2: "Durchwandern" der Stadt C3: Unterschiedliches Verschieben im Landscape & Application Modus
3	Rift	B3.2: AVATARS WOULD BE NICE B3.5: APPLICATION NAMES NOT ALWAYS READABLE
4	Vive	C1: red X to close application closer to the application and maybe smaller move landscape like the application spectate menu closable while spectating
4	Rift	B1.5: showing with lasers more intuitive B2.2: up to ca. 4 people B2.3: after a little B3.2: Give them a body B4.1: after a little C1: Schließen Button nach oben links Drehen mit Joystick dicker Laser (mit Laserschwertgeräusch beim kreuzen) Strand Umgebung Wasser \ Aqua
5	Vive	B3.1: spectator menü sollte man schließen können B4.1: spectator modus ist überflüssig, wenn man sich auch so etw. zeigen kann C2: - Klassen ausschreiben, nicht "impl..." - größere Schriften für kleine Klassen
5	Rift	
6	Vive	B1.5: Mehrfach highlighten noch besser

6	Rift	B1.4: Navigation mit Joystick ist einfacher B3.5: Brille war ein wenig unscharf B3.6: Zusätzliche Infos zu Klassen an Controller umständlich
7	Vive	C1: zoom in/out um text einer applikation usw. besser sehen zu können. momentan ruder ich sehr viel mit den Armen, um mir etwas ranzuziehen. (wahrscheinlich durch Teleport aber eigentlich gelöst) C2: Lag / Bilschirm "flackern" bei schließen von Komponenten sollte verbessert werden
7	Rift	B1.1: Genau wie Applikationen bewegen ist viel intuitiver B2.2: Ein Leiter, der verschieben kann, und Zuschauer, die Highlights, könnte bei mehr Leuten besser sein B2.3: Es wirkte so, als sei die Framerate instabil B3.6: Nicht wirklich aufgefallen B4.2: Hat keine wirklich nützliche Funktion C1: Darstellung als Kugeln, um Verbindungen besser sichtbar zu machen, im Gegensatz zu jetziger 2-dimensionaler Darstellung C2: Neigung von Menü und Info-Fenster etwas zum User hin bewegen
8	Vive	B4.2: Ich weiß nicht, wie sinnvoll der Spectator-Mode insgesamt ist. Im Prinzip ist man in der VR als Beobachter/in schon dabei und fühlt sich sowieso wohler aus eigener Perspektive
8	Rift	
9	Vive	C2: Blauen Kasten früher anzeigen um Mobilität zu erleichtern
9	Rift	B2.2: Um etwas zu zeigen auf jeden Fall, Kommunikation ist aber schwierig B4.1: Ungewohntes Gefühl, aber nicht besonders schlimm B4.2: Jedoch nur mit dem Grund, da es sich etwas unangenehm anfühlt C2: Den Coop-Partner als Körper darstellen, wie z.B. die Hände im "Eröffnungs-Raum" der Oculus
10	Vive	B3.5: teilweise unscharf oder zu klein (liegt vielleicht auch an der Brille) B3.6: Vorschlag: Text vergrößern, wenn man drauf zeigt C3: sehr nette Arbeitsatmosphäre
10	Rift	B1.2: Es wäre hilfreich Objekte aus der Entfernung greifen zu können B1.3: Keine selbstdrehen, keine kontinuierliche Bewegung B1.4: Man braucht ein bisschen viele einzelne Klicks B2.2: Wenn man den Platz und die Geräte hat schon B2.3: No B3.5: Manche waren etwas klein B3.6: Manche Label wurden abgeschnitten wenn sie zu lang waren B4.1: No B4.2: Yes C1: kontinuierliche Bewegung Objekte aus der Entfernung greifen Eine Suchfunktion C2: kleine labels an den Controllerbuttons (in VR) die sagen was der Button bewirkt
11	Vive	B1.1: Rotation fehlt C1: Personalisierung d. Hintergrunds, Menüfarben etc. C2: Highlights verschwinden bei Bewegung C3: Hat Spaß gemacht!

11	Rift	<p>B4.1: eye strain</p> <p>C1: animated connections to symbolize traffic or usage</p> <p>C2: spectator mode: highlighting should stay the same if the application is moved; applications should be able to be grabbed through the ground (or you shouldn't be able to go through the ground)</p>
----	------	--